PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: Takeshi OGASAWARA et al.

Group Art Unit: 2171

Serial No.: 10/621,281

Docket: 728-233
(JP9-2001-0303US1)

Filed: July 16, 2003

Dated: April 22, 2004

For: METHOD AND APPARATUS FOR PRECISION OPTIMIZATION IN COMPILED PROGRAMS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF PRIORITY DOCUMENT

Sir:

Enclosed is a certified copy of Japanese Appln. No. 2002-224207 filed on July 31, 2002, from which priority is claimed under 35 U.S.C. §119.

Respectfully submitted,

Paul J. Farrell
Registration No. 33,494
Attorney for Applicants

DILWORTH & BARRESE, LLP
333 Earle Ovington Boulevard
Uniondale, New York 11553
(516) 228-8484

CERTIFICATE OF MAILING UNDER 37 C.F.R. § 1.8 (a)

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the: Commissioner of Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on April 22, 2004.

Dated: April 22, 2004

Paul J. Farrell

日　本　国　特　許　庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日
Date of Application: ２００２年　７月３１日

出願番号
Application Number: 特願２００２－２２４２０７

[ST.10/C]: [ＪＰ２００２－２２４２０７]

出願人
Applicant(s): インターナショナル・ビジネス・マシーンズ・コーポレーション

２００３年　２月　７日

特許庁長官
Commissioner,
Japan Patent Office 太田信一郎

出証番号　出証特２００３－３００５３５３

【書類名】　特許願

【整理番号】　JP9010303

【提出日】　平成14年 7月31日

【あて先】　特許庁長官　殿

【国際特許分類】　G06F　9/45

【発明者】

【住所又は居所】　神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内

【氏名】　菅沼　俊夫

【発明者】

【住所又は居所】　神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　東京基礎研究所内

【氏名】　小笠原　武史

【特許出願人】

【識別番号】　390009531

【氏名又は名称】　インターナショナル・ビジネス・マシーンズ・コーポレーション

【代理人】

【識別番号】　100086243

【弁理士】

【氏名又は名称】　坂口　博

【代理人】

【識別番号】　100091568

【弁理士】

【氏名又は名称】　市位　嘉宏

【代理人】

【識別番号】　100108501

【弁理士】

【氏名又は名称】　上野　剛史

【復代理人】

【識別番号】　100104880

【弁理士】

【氏名又は名称】　古部　次郎

【手数料の表示】

【予納台帳番号】　081504

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　9706050

【包括委任状番号】　9704733

【包括委任状番号】　0207860

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　プログラム変換方法、これを用いたデータ処理装置及びプログラム

【特許請求の範囲】

【請求項１】　コンピュータを制御して、プログラムをコンパイルして機械語コードを生成するプログラム変換方法において、

前記プログラムを構成するメソッドを実行した際に、当該メソッドが浮動小数点演算における倍精度モードと単精度モードのどちらの状態で呼ばれたかについての情報をメモリに格納する第１のステップと、

前記プログラムにおいて浮動小数点演算における倍精度モードと単精度モードのうちの一方を基準とした場合に、所定のメソッドのコンパイル時に、当該メソッドが基準ではない精度モードで呼ばれた頻度を前記メモリに格納されている情報に基づいて調べる第２のステップと、

得られた前記基準ではない精度モードで呼ばれた頻度の情報に基づいて、前記所定のメソッドに関して、当該基準ではない精度モードでの呼び出しに対応した専用の機械語コードを生成し、メモリに格納する第３のステップと

を含むことを特徴とするプログラム変換方法。

【請求項２】　前記第１のステップでは、前記プログラムのランタイムルーチンにより、前記メソッドが前記基準ではない精度モードで呼ばれた回数を計数してメモリに格納し、

前記第２のステップでは、前記メモリに格納されている計数値と予め設定された閾値とを比較し、

前記第３のステップでは、前記計数値が前記閾値を超えた場合に前記専用の機械語コードを生成することを特徴とする請求項１に記載のプログラム変換方法。

【請求項３】　前記第３のステップでは、コンパイルしようとする前記所定のメソッドに関して、当該メソッドを前記基準ではない精度モードで動作させた方が処理コストを削減させ得る場合に、当該メソッドに対する前記専用の機械語コードを生成することを特徴とする請求項１に記載のプログラム変換方法。

【請求項４】　前記第３のステップでは、コンパイルしようとする前記所定

のメソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換えた後に、前記専用の機械語コードを生成することを特徴とする請求項１に記載のプログラム変換方法。

【請求項５】　前記第３のステップでは、コンパイルしようとする前記所定のメソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分けた後に、前記専用の機械語コードを生成することを特徴とする請求項１に記載のプログラム変換方法。

【請求項６】　コンピュータを制御して、プログラムをコンパイルして機械語コードを生成するプログラム変換方法において、

コンパイルしようとする対象メソッドに関し、当該対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成し、メモリに格納する第１のステップと、

前記対象メソッドと前記呼び出し側メソッドとの関係を調べる第２のステップと、

得られた関係に応じて、補助的なコードを生成し、前記メモリに格納されているコードに加える第３のステップと

を含むことを特徴とするプログラム変換方法。

【請求項７】　前記第２のステップでは、前記呼び出し側メソッドが倍精度モードまたは単精度モードのどちらであるかを調べ、

前記第３のステップでは、前記呼び出し側メソッドにおける精度モードと前記対象メソッドにおける精度モードとが異なる場合に前記補助的なコードを生成し追加することを特徴とする請求項６に記載のプログラム変換方法。

【請求項８】　前記第１のステップでは、コンパイルしようとする前記対象メソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換えた後に、機械語コードを生成することを特徴とする請求項６に記載のプログラム変換方法。

【請求項９】　前記第１のステップでは、コンパイルしようとする前記対象メソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分けた後に、前記領域ごとに演算精度を設定して機械語

コードを生成することを特徴とする請求項６に記載のプログラム変換方法。

【請求項１０】　プログラムをメソッド単位でコンパイルするコンパイラと

プログラムをインタプリタで実行すると共に前記コンパイラにてコンパイルされたコードで実行するプログラム実行部とを備え、

前記コンパイラは、浮動小数点演算における倍精度モードと単精度モードのうちの基準となる一方の精度モードで前記メソッドの第１のコードを生成すると共に、当該メソッドが一定の条件を満足する場合に、倍精度モードと単精度モードのうちの基準ではない精度モードで前記メソッドの第２のコードを生成することを特徴とするデータ処理装置。

【請求項１１】　前記プログラム実行部は、前記基準ではない精度モードで動作する前記メソッドから前記基準ではない精度モードで動作する他のメソッドを呼び出す場合に、前記コンパイラにて生成された前記第２のコードを呼び出すことを特徴とする請求項１０に記載のデータ処理装置。

【請求項１２】　前記コンパイラは、コンパイルしようとするメソッドが、前記第２のコードの方が処理コストを削減することができ、かつ当該メソッドが前記基準ではない精度モードのメソッドから呼ばれる頻度が高い場合に、前記第２のコードを生成することを特徴とする請求項１０に記載のデータ処理装置。

【請求項１３】　前記コンパイラは、

コンパイルしようとするメソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換える過剰精度最適化部と、

前記過剰精度最適化部による処理結果を反映させて前記第１のコードまたは前記第２のコードを生成するコード生成部と

を備えることを特徴とする請求項１０に記載のデータ処理装置。

【請求項１４】　前記コンパイラは、

コンパイルしようとするメソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分ける精度領域解析部と、

前記精度領域解析部による解析結果に基づき、前記領域ごとに演算精度を設定して機械語コードを生成するコード生成部と

を備えることを特徴とする請求項１０に記載のデータ処理装置。

【請求項１５】　プログラムをコンパイルして機械語コードを生成するデータ処理装置において、

コンパイルしようとする対象メソッドに関し、当該対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成するコード生成手段と、

前記対象メソッドの精度モードと前記呼び出し側メソッドの精度モードとの関係に基づいて、当該精度モードを合わせるための補助的なコードを生成し、前記コード生成手段にて生成された機械語コードに追加する補助コード追加手段と

を備えることを特徴とするデータ処理装置。

【請求項１６】　前記補助コード追加手段は、前記対象メソッドにおける精度モードと前記呼び出し側メソッドにおける精度モードとが異なる場合に、呼び出し側メソッドに対して精度モードを合わせるためのコードを追加することを特徴とする請求項１５に記載のデータ処理装置。

【請求項１７】　コンパイルしようとする対象メソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換えるコード編集手段をさらに備え、

前記コード生成手段は、前記コード編集手段による処理結果を反映させて機械語コードを生成することを特徴とする請求項１５に記載のデータ処理装置。

【請求項１８】　コンパイルしようとする対象メソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分ける精度領域解析手段をさらに備え、

前記コード生成手段は、前記精度領域解析手段による解析結果に基づき、前記領域ごとに演算精度を設定して機械語コードを生成することを特徴とする請求項１５に記載のデータ処理装置。

【請求項１９】　コンピュータを制御して、実行プログラムをコンパイルして機械語コードを生成するプログラムであって、

前記実行プログラムを構成するメソッドを実行した際に、当該メソッドが浮動小数点演算における倍精度モードと単精度モードのどちらの状態で呼ばれたかに

ついての情報をメモリに格納する第１の処理と、

前記実行プログラムにおいて浮動小数点演算における倍精度モードと単精度モードのうちの一方を基準とした場合に、所定のメソッドのコンパイル時に、当該メソッドが基準ではない精度モードで呼ばれた頻度を前記メモリに格納されている情報に基づいて調べる第２の処理と、

得られた前記基準ではない精度モードで呼ばれた頻度の情報に基づいて、前記所定のメソッドに関して、当該基準ではない精度モードでの呼び出しに対応した専用の機械語コードを生成し、メモリに格納する第３の処理と

を前記コンピュータに実行させることを特徴とするプログラム。

【請求項２０】　前記プログラムによる前記第３の処理では、コンパイルしようとする前記所定のメソッドに関して、当該メソッドを前記基準ではない精度モードで動作させた方が処理コストが低下する場合に、当該メソッドに対する前記専用の機械語コードを前記コンピュータに生成させることを特徴とする請求項１９に記載のプログラム。

【請求項２１】　前記第３の処理では、コンパイルしようとする前記所定のメソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換えた後に、前記専用の機械語コードを前記コンピュータに生成させることを特徴とする請求項１９に記載のプログラム。

【請求項２２】　前記第３の処理では、コンパイルしようとする前記所定のメソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分けた後に、前記専用の機械語コードを前記コンピュータに生成させることを特徴とする請求項１９に記載のプログラム。

【請求項２３】　コンピュータを制御して、実行プログラムをコンパイルして機械語コードを生成するプログラムであって、

コンパイルしようとする対象メソッドに関し、当該対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成し、メモリに格納する第１の処理と、

前記対象メソッドと前記呼び出し側メソッドとの関係を調べる第２の処理と、

得られた関係に応じて、補助的なコードを生成し、前記メモリに格納されてい

るコードに加える第３の処理と

を前記コンピュータに実行させることを特徴とするプログラム。

【請求項２４】　前記プログラムによる前記第３の処理は、前記対象メソッドにおける精度モードと前記呼び出し側メソッドにおける精度モードとが異なる場合に、呼び出し側のメソッドに対して精度モードを合わせるためのコードを前記コンピュータに追加させることを特徴とする請求項２３に記載のプログラム。

【請求項２５】　前記第１の処理に先だって、コンパイルしようとする前記対象メソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換える処理を、前記コンピュータにさらに実行させることを特徴とする請求項２３に記載のプログラム。

【請求項２６】　前記第１の処理に先だって、コンパイルしようとする前記対象メソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分ける処理を、前記コンピュータにさらに実行させると共に、

前記第１のステップでは、前記コンピュータに、前記領域ごとに演算精度を設定して機械語コードを生成させることを特徴とする請求項２３に記載のプログラム。

【請求項２７】　コンピュータを制御して実行プログラムをコンパイルして機械語コードを生成するプログラムを、当該コンピュータが読み取り可能に記録した記録媒体であって、

前記プログラムは、

前記実行プログラムを構成するメソッドを実行した際に、当該メソッドが浮動小数点演算における倍精度モードと単精度モードのどちらの状態で呼ばれたかについての情報をメモリに格納する第１の処理と、

前記実行プログラムにおいて浮動小数点演算における倍精度モードと単精度モードのうちの一方を基準とした場合に、所定のメソッドのコンパイル時に、当該メソッドが基準ではない精度モードで呼ばれた頻度を前記メモリに格納されている情報に基づいて調べる第２の処理と、

得られた前記基準ではない精度モードで呼ばれた頻度の情報に基づいて、前記

所定のメソッドに関して、当該基準ではない精度モードでの呼び出しに対応した専用の機械語コードを生成し、メモリに格納する第３の処理と

を前記コンピュータに実行させることを特徴とする記録媒体。

【請求項２８】　コンピュータを制御して実行プログラムをコンパイルして機械語コードを生成するプログラムを、当該コンピュータが読み取り可能に記録した記録媒体であって、

前記プログラムは、

コンパイルしようとする対象メソッドに関し、当該対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成し、メモリに格納する第１の処理と、

前記対象メソッドと前記呼び出し側メソッドとの関係を調べる第２の処理と、

得られた関係に応じて、補助的なコードを生成し、前記メモリに格納されているコードに加える第３の処理と

を前記コンピュータに実行させることを特徴とする記録媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、プログラムのコンパイル技術に関し、特に浮動小数点演算の精度保証を効果的に行う方法に関する。

【０００２】

【従来の技術】

Ｊａｖａ（サン・マイクロシステムズ・インコーポレーテッドの商標）の浮動小数点演算は、単精度演算および倍精度演算それぞれについて精度が言語仕様によって定められている（単精度演算は仮数部２４ビット・指数部８ビット、倍精度演算は仮数部５３ビット・指数部１１ビット）。

一方、ＩＡ３２（Intel Architecture-32）アーキテクチャのＣＰＵ（いわゆる米国インテル社のｘ８６ファミリ及び互換のＣＰＵ）は、この単精度演算及び倍精度演算を行うために、

（方法１）浮動小数点コントロールワード（以下ＦＣＷ）の浮動小数点演算モー

ド設定でＣＰＵのモードを切り替えるか、

（方法２）倍精度モードで演算を行い、単精度の演算結果を得る場合には、浮動小数点レジスタの値をメモリへ単精度でストアし読み出すことにより演算精度を落とす、

といった手法をとる必要があった。

【０００３】

ここで、例として単精度演算を行うメソッドを考える。一般にこのメソッドは、倍精度演算のＣＰＵモード（以下、倍精度モード）で呼ばれる場合と、単精度演算のＣＰＵモード（以下、単精度モード）で呼ばれる場合のどちらもあり得る。単精度モードで呼ばれる場合は単精度モードを仮定したコードを実行すれば効率がよい。倍精度で呼ばれる場合は、上述した方法１または方法２により演算の精度保証を行う必要がある。効率の良い方法としては、プログラムを解析して方法１を用いた場合と方法２を用いた場合の処理コストを比較し、より高速な方（オーバーヘッドの少ない方）を実行することができる。

【０００４】

ＪａｖａにおけるＪＩＴコンパイラなどのダイナミックコンパイラで、実行頻度の高いメソッド等を部分的にコンパイルするような分割コンパイル環境の場合、メソッドをコンパイルする際に、当該メソッドが単精度モードと倍精度モードのどちらで呼ばれるかはわからないので、例えばメソッド間の受け渡しは倍精度モードに固定し（すなわち全てのメソッドは倍精度モードで呼ばれる）、単精度演算を行う場合には上記のコスト計算に応じてコード生成を行う。

この種の従来技術としては、例えば下記文献に開示された技術がある。

文献：The Java HotSpot Server Compiler, by M.Paleczny, C.Vick, and C.Click. In Proceeding of USENIX Java Virtual Machine Research and Technology Symposium (JVM '01)

【０００５】

【発明が解決しようとする課題】

しかしながら、上記従来の技術は、方法１、２のいずれも処理コストが大きく、多大なオーバーヘッドがかかるという欠点があった。

具体的には、方法１の場合、ＦＣＷの切り替え命令（fldcw word ptr [mem]）がコストの高い命令である。また、ＦＣＷをいったん切り替えると、その範囲内でのメソッド呼び出しの前後では再びＦＣＷの切り替えを行い、基準である倍精度モードに戻す必要があるため、さらなるオーバーヘッドの原因となる。

方法２の場合、単精度演算のたびに、メモリへの書き込み読み出し命令を実行することによってオーバーヘッドを生じ、実行性能の低下を招く。

【０００６】

また、そもそもダイナミックコンパイラにおいて、メソッド間解析は、コンパイル時間その他の理由により制限されるため、有効な方法とはいえない。またダイナミックに新しいクラスがロードされるため、いったん解析した結果が無効になるといった無駄が生じる。

そこで、本発明は、浮動小数点演算における単精度演算と倍精度演算の精度保証を効率的に行い、オーバーヘッドや解析処理の無駄の発生を回避することを目的とする。

【０００７】

【課題を解決するための手段】

上記の目的を達成するため、本発明は、コンピュータを制御して、プログラムをコンパイルして機械語コードを生成する、次のようなプログラム変換方法として実現することができる。すなわち、プログラムを構成するメソッドを実行した際に、このメソッドが浮動小数点演算における倍精度モードと単精度モードのどちらの状態で呼ばれたかについての情報をメモリに格納する第１のステップと、このプログラムにおいて浮動小数点演算における倍精度モードと単精度モードのうちの一方を基準とした場合に、所定のメソッドのコンパイル時に、このメソッドが基準ではない精度モードで呼ばれた頻度をメモリに格納されている情報に基づいて調べる第２のステップと、得られた基準ではない精度モードで呼ばれた頻度の情報に基づいて、かかる所定のメソッドに関して、基準ではない精度モードでの呼び出しに対応した専用の機械語コードを生成し、メモリに格納する第３の

ステップとを含むことを特徴とする。

【０００８】

なお、Ｊａｖａのようにメソッド間の受け渡しを倍精度モードで行うことが基本となっているシステムでは、基準ではない精度モードは単精度モードとなる。

基準ではない精度モードでの呼び出しの頻度は、例えば、プログラムの実行時に基準ではない精度モードで呼び出された回数を計数しておき、計数値と所定の閾値とを比較することにより判断することができる。

また、専用の機械語コードを生成するのは、コンパイル対象であるメソッドを基準ではない精度モードで動作させた方が処理コストを削減させ得る場合であることが好ましいことは言うまでもない。

さらに、専用の機械語コードを生成する場合に、前処理として、コンパイル対象であるメソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換える処理を行うことにより、専用の機械語コードを生成する対象メソッドの数が多くなる。あるいは前処理として、コンパイル対象のメソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分ける処理を行うことにより、より精度モードの適切なコード生成を行うことができる。

【０００９】

本発明による他のプログラム変換方法は、コンパイルしようとする対象メソッドに関し、この対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成し、メモリに格納する第１のステップと、この対象メソッドと呼び出し側メソッドとの関係を調べる第２のステップと、得られた関係に応じて、補助的なコードを生成し、メモリに格納されているコードに加える第３のステップとを含むことを特徴とする。

【００１０】

ここで、詳細には、第２のステップでは、呼び出し側メソッドが倍精度モードまたは単精度モードのどちらであるかを調べ、第３のステップでは、呼び出し側メソッドにおける精度モードと対象メソッドにおける精度モードとが異なる場合

に精度モードを合わせるための補助的なコードを生成し追加する。

さらに、専用の機械語コードを生成する場合に、前処理として、コンパイル対象であるメソッド中から過剰な精度で演算が実行される部分を検出し、単精度の演算に置き換える処理を行うことにより、専用の機械語コードを生成する対象メソッドの数が多くなる。あるいは前処理として、コンパイル対象のメソッドに対してコード解析を行い、単精度による演算を行う領域と、倍精度による演算を行う領域とに分ける処理を行うことにより、メソッド単位よりもさらに細かいレベルでプログラムの部分ごとに最適なコード生成を行うことができる。

【００１１】

また、上記の目的を達成する他の本発明は、次のように構成されたデータ処理装置としても実現される。すなわち、このデータ処理装置は、プログラムをメソッド単位でコンパイルするコンパイラと、プログラムをインタプリタで実行すると共にコンパイラにてコンパイルされたコードで実行するプログラム実行部とを備える。このコンパイラは、浮動小数点演算における倍精度モードと単精度モードのうちの基準となる一方の精度モードでメソッドの第１のコードを生成すると共に、このメソッドが一定の条件を満足する場合に、倍精度モードと単精度モードのうちの基準ではない精度モードでメソッドの第２のコードを生成することを特徴とする。

【００１２】

さらに、本発明は、コンパイルしようとする対象メソッドに関し、この対象メソッドを呼び出す呼び出し側メソッドの浮動小数点演算における精度モードに応じて演算精度を設定し、機械語コードを生成するコード生成手段と、この対象メソッドの精度モードと呼び出し側メソッドの精度モードとの関係に基づいて、精度モードを合わせるための補助的なコードを生成し、コード生成手段にて生成された機械語コードに追加する補助コード追加手段とを備えるデータ処理装置としても実現される。

【００１３】

また、本発明は、コンピュータを制御して上述したプログラム変換方法における各ステップに相当する処理を実行させるプログラムや、上述したデータ処理装

置としてコンピュータを機能させるプログラムとしても実現される。このプログラムは、磁気ディスクや光ディスク、半導体メモリ、その他の記録媒体に格納して配布したり、ネットワークを介して配信したりすることにより提供することができる。

【００１４】

【発明の実施の形態】

以下、添付図面に示す実施の形態に基づいて、この発明を詳細に説明する。

本発明では、コンパイル時のコード生成において、浮動小数点演算における精度保証を行う上で、ＦＣＷ（浮動小数点コントロールワード）のモード設定の切り替えやメモリへの読み書きといったオーバーヘッドを削減するために、次の２つの手法を提案する。すなわち、原則的には単精度モードと倍精度モードのうちの一方を基準としてコード生成を行い、プログラムを実際に動作させて、基準ではないモードで呼ばれることの多いメソッドについて特殊化（specialization）により、他方のモードに対応したコード生成を行う手法（第１の実施の形態）と、呼び出し側（コンパイル対象のメソッドを呼び出したメソッド）の精度モードに従って、当該メソッド単体で最適化するようにコード生成を行う手法（第２の実施の形態）である。

また、本発明では、上記の２つの手法をさらに効果的に実現するため、これらのコード生成の前処理として、メソッド内の演算を置き換える手法（第３の実施の形態）、及びメソッド内のコードを解析する手法（第４の実施の形態）を提案する。

なお、以下の実施の形態では、ＪａｖａのＪＩＴコンパイラに適用する場合を例として説明する。すばわち、メソッド間の受け渡しは倍精度モードを基準とし、単精度モードでのメソッド呼び出しを取り得る場合に、プログラムの特殊化を行うこととする。

【００１５】

〔第１の実施の形態〕

第１の実施の形態では、実際にプログラムを走らせた上で、所定のメソッドをコンパイルする場合に、当該メソッドについて単精度モードを基準とし、かつ単

精度モードを基準とする他のメソッドからの呼び出しが十分多い場合に、当該メソッドについて特殊化を行う。これにより、ＦＣＷの切り替えや精度保証のためのメモリ書き込み読み出し命令のない、効率のよいコードの生成を実現する。

【００１６】

図１は、第１の実施の形態による浮動小数点演算の精度保証方法が実現されるコンピュータシステムの構成を説明する図である。

図１を参照すると、本実施の形態におけるコンピュータシステムは、ソースプログラム（入力コード）をコンパイルするコンパイラ１００と、コンパイラ１００にてコンパイルされたオブジェクトプログラム（出力コード）を実行して種々の処理を行うプログラム実行部２００と、メモリ３００とを備える。コンパイラ１００及びプログラム実行部２００は、パーソナルコンピュータやワークステーションなどのコンピュータシステムにおけるプログラム制御されたＣＰＵにて実現される。メモリ３００は、コンピュータ装置のメインメモリであり、ＲＡＭ等で実現される。メモリ３００には、ＣＰＵを制御してコンパイラ１００として動作させるためのプログラムやコンパイルの対象であるプログラム（以下、実行プログラムと称す）が格納されると共に、この実行プログラムを実行する際に用いられるランタイムルーチン３１０が格納される。なお、メモリ３００に格納されるプログラムは、必要に応じて、適宜磁気ディスクその他の外部記録装置に保存されることは言うまでもない。

【００１７】

図１において、コンパイラ１００は、バイトコード（byte code）で記述された入力コードを入力して処理し、機械語で記述された出力コードを生成して出力する。この入力コードの入力は、コード生成装置４００にて生成された入力コードを直接入力したり、コード生成装置４００にて生成された入力コードを記憶した記憶装置５００から入力したり、ネットワーク６００上に存在するコード生成装置４００や記憶装置５００からネットワーク６００を介して入力したりすることにより行われる。コンパイラ１００により生成された出力コードは、プログラム実行部２００により実行される。プログラム実行部２００は、コンパイラ１００にてコンパイルされた実行プログラムのオブジェクトコード（機械語コード）

を実行するほか、当該実行プログラムをインタプリタによっても実行する。そして、実行プログラムを実行した際に、その実行プロファイルを取得してメモリ３００に蓄積する。

【００１８】

以下、図２乃至図５を参照して、本実施の形態による操作を詳細に説明する。この操作は、メモリ３００に格納されたランタイムルーチン３１０にて実現される。

図２乃至図４は、本実施の形態による操作を実現するための、ランタイムルーチンにより生成されるデータ構造を示す図であり、図２は初期状態を示し、図３は最初のコンパイル時の典型的なデータ構造を示し、図４は特殊化が実行された場合のデータ構造を示す。また、図５は、これらと比較するための従来のランタイムルーチンにより生成されるデータ構造を示す図である。

【００１９】

図２、５に示すように、プログラムを構成する所定のメソッド１１、１２を実行してメソッド呼び出し（図ではメソッドコールと表記）「call target」を実行する場合、プログラム実行部２００は、まずメソッドブロック２０の対応するエントリを参照する。そして、当該エントリに登録された、呼び出そうとする対象のメソッド（以下、対象メソッドと称す）のコードアドレスの登録ポイントから、このエントリに対応するゲートコード３１と必要な他のランタイムルーチン４０とを経て、呼び出した対象メソッドをインタプリタまたはコンパイルされたコードで実行する。対象メソッドを実行した後は、呼び出し元であるメソッド１１、１２に戻る。

【００２０】

本実施の形態におけるデータ構造を示す図２と従来のデータ構造を示す図５とを比較すると、本実施の形態では、メソッドブロック２０における対象メソッドのコードアドレスの登録ポイントとして、上述した通常のエントリ２１（Code Ptr1）の他に単精度専用のエントリ２２（Code Ptr2）が設けられている。そして、呼び出し側メソッド１１、１２のコールサイトにおける現在のＦＣＷのモードによって、通常のエントリ２１または単精度専用のエントリ２２を呼び分ける。

すなわち、単精度モードを基準とするメソッド１２から対象メソッドを呼び出す場合、この新たに設けられた単精度専用のエントリ２２を参照することとなる。

また、このエントリ２２に対応するゲートコード３２として、

１）ＦＣＷを倍精度に設定し（fldcw f2d）、

２）専用カウンタ（単精度モードカウンタ）をインクリメントし（inc counter）、

３）通常のゲートコードエントリへジャンプする（jmp Gate）

コードを用意する。

【００２１】

また、単精度モードでのメソッド呼び出しでは、単精度専用コードへジャンプすることを保証しない。すなわち、呼び出した対象メソッドが倍精度モードを基準とするメソッドである可能性がある。そこで、メソッド１２に示すように、呼び出し命令（call target）の直後、すなわち呼び出した対象メソッドの実行後に戻るアドレスに、ＦＣＷを単精度モードへ再設定する命令（fldcw d2f）を生成する。対象メソッドが倍精度モードと単精度モードのどちらの状態で呼ばれたかについての情報を取得する精度モード情報取得手段としての単精度モードカウンタの値は、メモリ３００に保持され、後述するコンパイル時にコンパイラ１００により参照される。

【００２２】

図５を参照すると、単精度モードでのメソッド呼び出しでは、メソッド間の受け渡しを倍精度モードに固定する原則にしたがって、メソッド１２に示すように、呼び出し命令（call target）の前でＦＣＷを倍精度モードに切り替え（fldcw f2d）呼び出し命令（call target）の後ろで単精度モードに戻していることがわかる（fldcw d2f）。

図２に示した状態では、単精度モードの下でのメソッド呼び出しは、ゲートコード３２によって、従来と同様にメソッドの境界で倍精度モードへと設定され、通常のパスを経て呼び出しポイントに戻ってきた後、再度、単精度モードへ再設定される。しかし、この状態のままでは、単精度モードからの呼び出しのパスには、ゲートコード３２におけるカウンタのインクリメントやジャンプ命令などの

オーバーヘッドがかかることになる。

【００２３】

次に、メソッド１１、１２から呼び出されるメソッドがコンパイルされる場合を考える。プログラムの実行が繰り返され、所定のメソッドの実行回数を計数するカウンタ値が予め定められた閾値を超えた場合に、コード生成手段であるコンパイラ１００により当該対象メソッドがコンパイルされてオブジェクトコードが生成され、当該対象メソッドの実行がインタプリタにおける実行からコンパイラにおける実行へ遷移する。

コンパイラ１００によるコンパイル時に、メモリ３００に保持されている単精度モードカウンタの値をチェックし、インタプリタからコンパイラへ遷移するためのカウンタ値と比較することによって、呼び出し側におけるＦＣＷのモードは単精度と倍精度のどちらの設定が多いかを知ることができる。

単精度モードカウンタの値が小さい場合、メソッド境界は倍精度モードであると仮定した従来と同様のオブジェクトコード（general version）５０を生成する。図３は、この時のデータ構造を示している。

このオブジェクトコード５０は、メソッドブロック２０における通常のメソッドのコードアドレスのエントリ２１に登録される。また、単精度モード専用のエントリ２２に対応したゲートコード３２内のジャンプアドレスのターゲットとなるように、このゲートコード３２におけるジャンプ命令が書き換えられる。

【００２４】

さらに、対象メソッドが実行時に頻繁に呼び出される（ホットな）メソッドである場合の再コンパイルを考える。

この対象メソッドの呼び出しに関して、単精度専用のエントリ２２に対応するゲートコード３２における単精度モードカウンタの値が十分大きい場合は、ＦＣＷが単精度モード設定の状態で（すなわちメソッド１２から）呼び出されたケースの頻度が高いと考えられる。このメソッドが単精度を基準モードとしてコンパイルする条件を満たす時、単精度モードでの直接呼び出しに特殊化されたバージョンのための再コンパイルを行うこととなる。かかる条件は任意に定め得るが、例えば、単精度モードでの呼び出し回数が倍精度モードでの呼び出し回数よりも

多い場合などというように、適当な閾値を設定することができる。図４は、再コンパイルにより、単精度モードでの直接呼び出しに用いられる特殊化されたオブジェクトコード（special version）６０を生成した場合のデータ構造を示している。

【００２５】

生成された特殊化されたオブジェクトコード６０は、メソッドブロック２０における単精度専用のコードアドレスのエントリ２２へ登録する。また、ゲートコード３２に対しては、この新しいオブジェクトコード６０へ直接ジャンプするように先頭コードを無条件ジャンプ命令に書き換える。これによって、単精度モードの仮想的なコールサイト（virtual call site）では、自動的に新しく登録されたコードを呼び出すことが可能となる。また、静的メソッド（static）や非仮想的なコールサイト（nonvirtual call site）で直接ゲートコード３２を呼んでいたところは、既存のバックパッチのメカニズムによって、次回の呼び出しから直接オブジェクトコード６０を呼び出すことになる。

最後に、この単精度モード専用のオブジェクトコード６０は、リターンアドレスのポインタを固定長バイト分だけスライドさせる。これによって、呼び出し側のメソッド１２における呼び出し命令（call target）の直後に置かれている、ＦＣＷを単精度モードに再設定するための命令（fldcw d2f）をスキップする。

【００２６】

上述したように、まず通常のコンパイルを行った後、メソッドの実行状態に応じて再コンパイルする際に特殊化されたオブジェクトコード６０を生成する手順の他に、図２に示した初期状態から初めて対象メソッドがコンパイルされる時に、単精度モードカウンタの値がすでに十分大きい時（例えばインタプリタにおける実行からコンパイラにおける実行へ遷移するための閾値の９０％を超える場合など）には、倍精度モードでの呼び出しを仮定した通常のオブジェクトコード５０を生成することなく、初めから単精度用の特殊化されたオブジェクトコード６０を生成するようにしても良い。この場合、通常のコンパイル（倍精度モードでの呼び出しを仮定したオブジェクトコード５０の生成）を後で駆動させる可能性を残すため、インタプリタにおける実行からコンパイラにおける実行への遷移の

ためのカウンタに対し、その値を調整する（例えば閾値からＦモードカウンタを引いた値に再設定する）ことが必要になる。

【００２７】

図６は、以上の図２乃至図４のデータ構造を参照して説明した、本実施の形態によるプログラムの特殊化の行程を示すフローチャートである。

図６に示すように、所定のメソッド（上記の説明における対象メソッド）の呼び出し頻度が十分に大きく（ホットメソッドを検出）、当該メソッドをコンパイルする場合、コンパイラ１００は、まずゲートコード３２の単精度モードカウンタの値が閾値を超えたかどうか（予め設定された条件を満たしたかどうか）を調べる（ステップ６０１、６０２）。

単精度モードカウンタの値が閾値を超えた場合は、次に当該メソッドが単精度モードを基準としてコンパイル可能かどうかを判断する（ステップ６０３）。これは、当該メソッド中の倍精度演算、単精度演算、メソッド呼び出し命令の数を数えて、当該メソッド全体について、単精度モードを基準とし必要に応じてＦＣＷを倍精度モードへ切り替えることと、倍精度モードを基準とし必要に応じてＦＣＷを単精度モードへ切り替える（またはメモリを用いた精度保証を行う）こととのどちらが低コストかを解析することによって判断できる。

【００２８】

処理対象であるメソッドが単精度モードを基準としてコンパイル可能と判断された場合、次にコンパイラ１００は、単精度モードでの呼び出しに特殊化されたオブジェクトコード６０を生成する（ステップ６０４）。そして、メソッドブロック２０の単精度専用コードアドレスのエントリ２２に登録し（ステップ６０５）、単精度専用のゲートコード３２において、fldcw f2d及びinc counterをnopに変換し（削除し）、ジャンプアドレスをオブジェクトコード６０のアドレスに書き換えて処理を終了する（ステップ６０６）。

【００２９】

一方、ステップ６０２で単精度モードカウンタの値が閾値を超えていない場合、またはステップ６０３で当該メソッドが単精度モードを基準としてコンパイル可能ではないと判断された場合、コンパイラ１００は、倍精度モードでの呼び出

しを仮定した（すなわち通常の）オブジェクトコード５０を生成し（ステップ６０７）、メソッドブロック２０における当該メソッドのコードアドレスのエントリ２１に登録する（ステップ６０８）。

次に、当該メソッドに関する単精度専用のオブジェクトコード６０が既に生成されているかどうかを調べ（ステップ６０９）、かかるオブジェクトコード６０が生成されていないならば、以後は新たに生成されたオブジェクトコード５０を使用するため、単精度専用のゲートコード３２におけるジャンプアドレスをゲートコード３１からオブジェクトコード５０のアドレスに書き換えて処理を終了する（ステップ６１０）。一方、単精度専用のオブジェクトコード６０が既に生成されているならば、単精度モードでの呼び出しに対しては当該オブジェクトコード６０を使用できるため、そのまま処理を終了する。

【００３０】

以上のように、第１の実施の形態によれば、倍精度モードを基準としつつも、単精度モードのメソッドからの呼び出し頻度の高いメソッドに対して、単精度専用の特殊化されたコードを生成する。そして、当該単精度モードのメソッドから当該特殊化されたコードを直接呼び出すことにより、ＦＣＷによる精度モードの切り替えやメモリへの書き込み読み出しといった、処理コストの高い冗長な処理を削減することができる。

また、プログラムの実行履歴に基づき、単精度モードのメソッドからの呼び出し頻度の高いメソッドに関して、この特殊化されたコードを生成するため、不必要なコードサイズの増加を招来することはない。

【００３１】

〔第２の実施の形態〕

第２の実施の形態では、コンパイル対象のメソッドは呼び出し側の精度モードで呼び出される。メソッドをコンパイルする際、コンパイラは、当該メソッドの開始位置と終了位置とにおける精度を、呼び出し精度と想定する。生成されたコードは想定された精度で実行開始され、終了する際には当該精度で終了するものとする。

第２の実施の形態による浮動小数点演算の精度保証方法は、第１の実施の形態

と同様に、図１に示すように構成されたコンピュータシステムにて実現される。

【００３２】

単精度の演算を行うプログラムでは、単精度演算を行うメソッド同士が呼び合うことが多い。そこで、単精度モードで実行するメソッドが別の単精度演算を行うメソッドを呼ぶ場合（この状況を、以下、対象状況と称す）、ＦＣＷによる精度モードの切り替えやメモリの書き込み読み出しを行わずに、そのまま実行できればオーバーヘッドを回避することができる。

Ｊａｖａのように、メソッド間の受け渡しにおいて倍精度モードを基準とする場合、対象状況においては、メソッドＡがメソッドＢを呼ぶ際の、

ａ）メソッドＡにおける単精度から倍精度へのモード変更、

ｂ）メソッドＢにおける倍精度から単精度へのモード変更、

及びメソッドＢがメソッドＡに戻る際の、

ｃ）メソッドＢにおける単精度から倍精度へのモード変更、

ｄ）メソッドＡにおける倍精度から単精度へのモード変更、

の４つの冗長なモード変換によるオーバーヘッドが、回避したいオーバーヘッドである。

本実施の形態では、上述したように、呼び出し側の精度モードでメソッドを呼ぶ。したがって、同一メソッドが複数の精度から呼び出される場合は、同一メソッドに対して精度に対応して複数のコードが生成されてしまう。コードサイズやコンパイル時間などでそれを抑制する必要がある場合、一度、いずれかの精度でメソッドがコンパイルされたら、それ以降は異なる精度で呼ぶ場合であっても、当該異なる精度に対応するコードを生成しないこともあり得る。その場合、精度を合わせるためのコードを経由してターゲットコードを呼ぶこととなる。また、この部分に関して上述した第１の実施の形態を適用しても良い。

【００３３】

上述したように本実施の形態において、コード生成手段であるコンパイラ１００は、メソッドをコンパイルする際、メソッドが呼ばれる時の精度モード（すなわちコールサイトの精度モード）で呼び出しが行われると想定し、コンパイル対象である当該メソッド単体を最も最適化するようにコード生成を行う。呼び出し

精度が倍精度ならば倍精度モードを仮定してコンパイルし（このコンパイルで生成されるコードを倍精度コードと称す）、呼び出し精度が単精度ならば単精度モードを仮定してコンパイルする（このコンパイルで生成されるコードを単精度コードと称す）。そして、倍精度演算と単精度演算とが混在するならば、精度モードの切り替え（方法１）やメモリへの書き込み読み出し（方法２）を用いた精度保証の手法を組み合わせてコンパイルする。

【００３４】

またコンパイラ１００は、コンパイルしたメソッドに関し、当該コードの入り口で仮定している精度モードに応じて、単精度コードか倍精度コードかの種別（デフォルト精度モード）を当該メソッドのコード中やランタイムルーチンに登録しておく。さらに、コンパイルしたメソッドに対して精度モードの切り替え（方法１）が行われた場合、精度モードが変化している区間とその精度モードを同様に登録する。

【００３５】

コンパイルしたコードが単精度コードしか存在せず、インタプリタで実行されるメソッドから呼ばれる可能性がある場合、インタプリタは常に倍精度モードで動いていると仮定し、呼び出しの前後で精度モードを切り替える。そして、当該コンパイルされたコードを実行する前に一旦単精度モードに切り替え、当該コードから戻ったときに再度倍精度モードに戻す。

【００３６】

所定のメソッドに着目するとき、このメソッドは静的メソッド（static method）である場合と仮想メソッド（virtual method）である場合とに分けられる。また、それぞれの場合で、メソッドに対して単精度コードが生成される場合と、倍精度コードが生成される場合とがある。当該メソッドが仮想メソッドでありかつ仮想的に呼び出される場合は、単精度コードから呼び出される場合と倍精度コードから呼び出される場合の両方があるため、当該メソッドに関して、単精度コードから呼び出された場合のコードと倍精度コードから呼び出された場合のコードとがそれぞれ生成され、プログラムの実行状況に応じて選択されることとなる。

【００３７】

次に、コンパイラ１００のコード生成について、具体的な場合に分けて、さらに説明する。

コンパイラ１００による分割コンパイル環境では、コンパイル対象であるメソッドと他のメソッドとの間で、次の２つの状況が生じ得る。

１：メソッドＡが既にコンパイルされて、今メソッドＢをコンパイルする。

２：メソッドＡがまだコンパイルされず、今メソッドＢをコンパイルする。

【００３８】

１の状況に対して、コンパイラ１００は次のようにコード生成を行う。

メソッドＡのコールサイトが倍精度コードである場合、メソッドＢの演算が倍精度演算のみならば倍精度でコンパイルし（倍精度コード）、単精度演算のみならば単精度でコンパイルする（単精度コード）。倍精度演算と単精度演算とが混在しているならば、処理コストを計算した上で倍精度コードまたは単精度コードでコンパイルし、必要に応じて精度モードの切り替えやメモリへの書き込み読み出しといった既存の手法により精度保証を行うか、または後に説明する第４の実施の形態を適用する。

一方、メソッドＡのコールサイトが単精度コードである場合、メソッドＢの演算が単精度演算のみならば単精度でコンパイルする（単精度コード）。それ以外ならば、倍精度コードでコンパイルし、必要に応じて精度モードの切り替えとメモリへの書き込み読み出しとを組み合わせる既存の手法により精度保証を行うか、または後に説明する第４の実施の形態を適用する。

図７は、以上の場合におけるコード生成を概念的に説明する図である。

【００３９】

２の状況に対して、コンパイラ１００は次のようにコード生成を行う。

メソッドＢの演算が倍精度演算のみならば倍精度でコンパイルし（倍精度コード）、単精度演算のみならば単精度でコンパイルする（単精度コード）。倍精度演算と単精度演算とが混在しているならば、処理コストを計算した上で倍精度コードまたは単精度コードでコンパイルし、必要に応じて精度モードの切り替えとメモリへの書き込み読み出しとを組み合わせる既存の手法により精度保証を行う

。倍精度コードと単精度コードのどちらを生成るかは、精度モードの切り替えとメモリへの書き込み読み出しとの組み合わせ方に依存する。または後に説明する第４の実施の形態を適用しても良い。

メソッドＡのコールサイトはインタプリタで実行されており、通常インタプリタは精度モードを決めて実行されるので、コンパイルコードを呼ぶためのglueコードで、必要に応じて精度モードを切り替える。すなわち、インタプリタが倍精度モードで実行されている場合、メソッドＢが倍精度コードであれば精度モードの切り替えを行わずにコンパイルされたコードを呼び出す。一方、メソッドＢが単精度コードであれば、glueコードにて倍精度モードから単精度モードに切り替えてコンパイルされたコードを呼び出し、メソッドＡに戻る際に倍精度モードに戻す。

図８は、以上の場合におけるコード生成を模式的に説明する図である。

【００４０】

図９は、上述したコンパイルの手順を説明するフローチャートである。

図９に示すように、コンパイラ１００は、ホットメソッドを検出して所定のメソッドをコンパイルする場合、まず、コンパイルしようとするメソッド（以下、対象メソッドと称す）内の演算の種類（単精度演算のみか、倍精度演算のみか、両者が混在するか）に応じてコードを生成し、メモリ３００に一時的に格納する（ステップ９０１、９０２）。次に、コンパイラ１００は、対象メソッドと当該対象メソッドを呼び出す他のメソッド（呼び出し側メソッド）との関係を検討し（ステップ９０３）、得られた関係に応じて、図７及び図８を参照して説明したように、補助的なコードを生成し、メモリ３００に格納されているステップ９０２で生成されたコードに加える。すなわち、必要な精度モード切り替えのためのコードを対象メソッドのコードや上記他のメソッドのコードに挿入したり、呼び出し側メソッドがコンパイルされていない場合はglueコードに挿入したりする（ステップ９０４）。

【００４１】

以上のように、第２の実施の形態によれば、所定のメソッドをコンパイルする場合に、呼び出し側（当該メソッドを呼び出したメソッド）の精度モードで開始

し終わることを前提とし、コンパイル対象であるメソッド単体で最適化されるようにコード生成を行う。そのため、所定の精度モード（例えば倍精度モード）を基準とすることによって発生する冗長な精度モードの切り替えを削減することができる。

【００４２】

〔第３の実施の形態〕

上記第１の実施の形態では、実際の動作において単精度で呼ばれることの多いメソッドに対して単精度を基準としたコードを生成し、第２の実施の形態では、メソッド内の演算の種類に応じて演算精度を設定してコードを生成した。しかしながら、プログラム中の個々のメソッドにおいては、単精度で十分な演算でありながら、倍精度で演算を行うように記述されているために倍精度モードでコードが生成されてしまうメソッドや、大部分が単精度の演算であるが、倍精度が必要な演算が部分的に現れるメソッドが存在する。このようなメソッドについては、第１、第２の実施の形態においても単精度モードでのコード生成を行うことができず、プログラムの実行効率の向上が抑制されてしまう。

そこで、第３の実施の形態は、メソッドをコンパイルする場合に、前処理としてメソッド内の演算を置き換えることにより、第１、第２の実施の形態によるコード生成の効果をさらに向上させる。

【００４３】

本実施の形態では、単精度で十分な演算でありながら、倍精度で演算を行うように記述されているために倍精度モードでコードが生成されてしまうメソッドを対象として、当該メソッド内の演算の置き換えを行う。このコードの編集処理を過剰精度最適化（XPO；excessive-precision optimization）と呼ぶ。

過剰な倍精度モードでコードが生成されてしまう原因としては、Ｊａｖａの提供するメソッドＡＰＩが倍精度の演算用の関数のみを提供したり、プログラマが過剰な精度を指定してメソッドを記述したりする等、様々な原因が考えられる。本実施の形態は、演算結果の精度ビット数を、関数の定義（definition）と使用（use）との関係（def-use関係）に基づいて伝搬する。そして、単精度演算で同じ結果を得られるならば、当該メソッド中の倍精度演算を単精度演算に置き換え

る。これにより、演算精度を落とすための無駄なメモリ書き込みを除去する。

【００４４】

第３の実施の形態による浮動小数点演算の精度保証のための過剰精度最適化は、第１の実施の形態と同様に、図１に示すように構成されたコンピュータシステムにて実現される。

図１０は、本実施の形態におけるコンパイラ１００の機能を説明するブロック図である。

図１０に示すように、本実施の形態のコンパイラ１００は、コンパイル対象のメソッドに対して過剰精度最適化の処理を行う過剰精度最適化部１１０と、過剰精度最適化処理を施されたメソッドのバイナリコードを機械語コードに変換するコード生成部１２０とを備える。過剰精度最適化部１１０及びコード生成部１２０は、メモリ３００に格納されているプログラムにて制御されたＣＰＵにて実現される仮想的なソフトウェアブロックである。なお図１０には、本実施の形態における特徴的な構成のみを記載している。実際には、図示の構成の他に、コンパイル対象のメソッドのバイナリコードを構文解析する手段や、本実施の形態による過剰精度最適化以外の種々の最適化処理を実行する手段が設けられることはいうまでもない。

【００４５】

コード編集手段である過剰精度最適化部１１０は、コード生成部１２０によるコード生成の前処理として、コンパイル対象のメソッド中から過剰精度（倍精度）の演算を行う部分を検出し、適切な低い精度（単精度）の演算に置き換える。具体的には、まずメソッド中の倍精度の演算（関数）に対して個別に過剰精度か否かの判定を行い、過剰精度と判断された演算を単精度に置き換える。ここで、コンパイル時にデータフロー解析を行い、検出された全ての倍精度による演算に対して機械的に過剰精度か否かの判定を行っても良いし、メソッドの実行によりメモリ３００等に蓄積される実行時プロファイルを参照して、頻繁に実行される倍精度の演算を対象として過剰精度か否かの判定を行っても良い。

【００４６】

ある浮動小数点計算Ｃは、次の２つの条件を満足する場合に、過剰精度計算で

あると判断される。すなわち、

（１）関数Ｆ（計算Ｃにおける入力を処理し出力を結果とするコード列、１命令コードの演算や別メソッドを呼び出す関数を含む）は、入力値がある制限された値（例えば仮数部を単精度で表現可能）である場合に、より悪い精度モード（単精度モード）で計算しても同じ結果を得られる。

（２）入力値は、全て上の制限された値である。

本実施の形態では、このような条件に合致する浮動小数点計算を集めた過剰精度計算表３２０を用意し、過剰精度最適化部１１０による過剰精度か否かの判定に用いる。図１１は、過剰精度計算表３２０の例を示す図である。図示のように、過剰精度計算表３２０には、上記条件（１）を満たす浮動小数点計算の種類（計算種別）ごとに、この条件（１）に合致するための入力値の制限が登録されている。この過剰精度計算表３２０は、過剰精度となることが予め経験的に分かっている浮動小数点計算について作成され、メモリ３００に格納されているものとする。

【００４７】

図１２は、過剰精度最適化部１１０が所定の計算Ｃに関して過剰精度か否かを判定する処理の流れを説明するフローチャートである。過剰精度最適化部１１０は、コンパイル対象であるメソッド中の各浮動小数点計算に対して、図１２に示す以下の処理を行う。

まず、所定の浮動小数点計算Ｃに着目する（ステップ１２０１）。そして、この計算Ｃにおける関数Ｆが過剰精度計算表３２０に登録されているかどうかを調べる（ステップ１２０２）。登録されているならば、次に、当該関数Ｆの入力値の全てが過剰精度計算表３２０に登録されている制限に該当するかどうかを判断する（ステップ１２０３）。該当するならば、着目中の計算Ｃは過剰精度計算であると判定する（ステップ１２０４）。一方、関数Ｆが過剰精度計算表３２０に登録されていないか、または関数Ｆの入力値のいずれかが過剰精度計算表３２０に登録されている制限に合致しない場合は、着目中の計算Ｃは過剰精度計算ではないと判定する（ステップ１２０２、１２０３、１２０５）。

【００４８】

図１３は、過剰精度計算と判定された計算Ｃを低精度（単精度）の演算に置き換える処理の流れを説明するフローチャートである。過剰精度最適化部１１０は、コンパイル対象であるメソッド中の各浮動小数点計算に対して、図１３に示す以下の処理を行う。

まず、着目中の計算Ｃが過剰精度計算か否かを判断する（ステップ１３０１）。これは図１２に示した処理による判定結果に基づく。計算Ｃが過剰精度計算であるならば、過剰精度最適化部１１０は、演算Ｆを低精度の演算に置換し、入力から精度変換の命令を削除し、定数を低精度で同じ値を表す定数表現に変換する（ステップ１３０２）。そして、入力に対する出力値の符号、仮数部、指数部の値がそれぞれどのような値を取り得るかに関する情報（以下、値情報）を生成し、演算Ｆ（特に出力を行うコード）に対して属性情報として登録する（ステップ１３０３）。

【００４９】

以上のようにして、過剰精度最適化部１１０による処理が終了する。この過剰精度最適化処理の結果、すなわち過剰精度計算の単精度化がなされたメソッドは、メモリ３００に保持され、コード生成部１２０による機械語コードの生成に用いられることとなる。

【００５０】

図１４及び図１５は、具体的な浮動小数点計算に関して過剰精度計算である倍精度計算を単精度計算に変換する例を示す図である。図１４は、単精度浮動小数点値に値０．５を乗算する倍精度計算を単精度化する例を疑似コードで示し、図１５は、単精度浮動小数点値の平方根sqrt(v)を倍精度で求めて単精度に丸める計算を単精度化する例を疑似コードで示す。

図１４において、ｆは単精度浮動小数点値を結果として残す任意の計算である。ＬＦ１、ＬＦ２、ＬＦ３は単精度浮動小数点値を持つ変数である。ＬＤ１、ＬＤ２、ＬＤ３は倍精度浮動小数点値を持つ変数である。Ｆ２Ｄは単精度浮動小数点値を倍精度浮動小数点値に変換する命令である。ＦＬＯＡＤは単精度浮動小数点の定数をロードする命令、ＤＬＯＡＤは倍精度浮動小数点の定数をロードする命令である。ＦＭＵＬは単精度浮動小数点の乗算する命令、ＤＭＵＬは倍精度浮

動小数点の乗算する命令である。

処理対象の倍精度計算において、変数ＬＤ２に命令Ｆ２Ｄと共に変数ＬＦ１が代入され、変数ＬＤ３に値０．５の倍精度浮動小数点値をロードする命令ＤＬＯＡＤが代入され、２つ目の変数ＬＤ２に倍精度の乗算命令ＤＭＵＬが代入されている。

【００５１】

この倍精度の浮動小数点計算に対して、過剰精度最適化部１１０による判定（図１２参照）が行われた場合を考える。このとき図１１に示した過剰精度計算表３２０が参照されるものとする。

すると、命令ＤＭＵＬは、過剰精度計算表３２０に登録されており、入力である変数ＬＤ２の実体が単精度浮動小数点値を持つ変数ＬＦ１であるので入力１の制限に該当する。そこで、過剰精度最適化部１１０はこの倍精度計算を過剰精度計算であると判定する。そして、倍精度計算を単精度計算に置き換える（図１３参照）。

図１４の例では、精度変換の命令ＬＤ２＝Ｆ２Ｄが削除され、命令ＤＬＯＡＤが値０．５の単精度浮動小数点値をロードする命令ＦＬＯＡＤに変換され（ＬＤ３→ＬＦ３）、さらに命令ＤＭＵＬが単精度の乗算命令ＦＭＵＬに変換される（ＬＤ２→ＬＦ２）。

以上のようにして、過剰精度最適化部１１０による単精度化（過剰精度最適化処理）の処理が終了する。

【００５２】

図１５において、ｆ、ＬＦ１、ＬＦ２、ＬＦ３、ＬＤ１、ＬＤ２、Ｆ２Ｄは、図１４の場合と同様である。Ｄ２Ｆは、倍精度浮動小数点値を単精度浮動小数点値に変換する命令である。ＦＳＱＲＴは単精度浮動小数点値の平方根を計算する命令、ＤＳＱＲＴは倍精度浮動小数点の平方根する命令である。

処理対象の倍精度計算において、１つ目の変数ＬＤ２に命令Ｆ２Ｄと共に変数ＬＦ１が代入され、２つ目の変数ＬＤ２に１つ目の変数ＬＤ２の倍精度浮動小数点値の平方根を求める命令ＤＳＱＲＴが代入され、変数ＬＦ３に２つ目の変数ＬＤ２の結果を単精度浮動小数点値に変換する命令Ｄ２Ｆが代入されている。

【００５３】

この倍精度の浮動小数点計算に対して、過剰精度最適化部１１０による判定（図１２参照）が行われた場合を考える。このとき図１１に示した過剰精度計算表３２０が参照されるものとする。

すると、命令ＤＳＱＲＴと命令Ｄ２Ｆとの組合せは、過剰精度計算表３２０に登録されており、入力である１つ目の変数ＬＤ２の実体が単精度浮動小数点値を持つ変数ＬＦ１であるので入力１の制限に該当する。そこで、過剰精度最適化部１１０はこの倍精度計算を過剰精度計算であると判定する。そして、倍精度計算を単精度計算に置き換える（図１３参照）。

図１５の例では、精度変換の命令ＬＤ２＝Ｆ２Ｄが削除され、命令ＤＳＱＲＴと命令Ｄ２Ｆとの組が単精度浮動小数点値の平方根を求める命令ＦＳＱＲＴに変換される（ＬＤ２＋ＬＦ３→ＬＦ３）。

以上のようにして、過剰精度最適化部１１０による単精度化（過剰精度最適化処理）の処理が終了する。

【００５４】

以上のようにして、本実施の形態ではメソッド中の過剰精度の計算を単精度の計算に置き換えるため、例えば第１の実施の形態と共に本実施の形態を用いれば、過剰精度の計算の単精度化により、単精度用の特殊化されたコードの生成対象となるメソッドが多くなる。また、第２の実施の形態と共に本実施の形態を用いれば、過剰精度の計算の単精度化により、当該メソッドにおいて、より精度モードの適切なコード生成を行うことが可能となる。

【００５５】

なお、本実施の形態では、過剰精度計算表３２０を予め作成してメモリ３００に格納しておくこととしたが、過剰精度計算表３２０のエントリを動的に追加していくことも可能である。すなわち、図１２を参照して説明した判定処理において、ステップ１２０２で所定の計算Ｃにおける関数Ｆが過剰精度計算表３２０に登録されていないと判断された場合に、当該計算Ｃが上述した過剰精度計算であるか否かを判断する条件（１）（２）に合致するか（入力値の全集合を使って計算Ｃから得た結果と、計算Ｃを低精度で実行して得た結果が同じであるか）を実

際に計算して調べ、合致するならば関数Ｆ及びその入力値を過剰精度計算表３２０に登録する。言い換えれば、メソッドの実行時に徐々に過剰精度計算表３２０を構築する方法である。なお、計算Ｃが条件（１）（２）に合致すると判断した場合、過剰精度計算表３２０に登録すると共に図１３に示した置き換え処理に移行しても良いし、過剰精度計算表３２０を作成する処理と図１２の判定処理とを別個の処理として、当該計算Ｃを対象として改めて図１２の判定処理を行っても良い。ただし、計算Ｃが条件（１）（２）に合致するかどうかを判断する処理は多大な時間（処理コスト）を要するため、特に重要な計算に限定して用いたり、この処理コストがプログラムの実行におけるオーバーヘッドとならない静的コンパイラに用いたりすることが好ましい。

【００５６】

〔第４の実施の形態〕

次に、第４の実施の形態は、単精度の演算と倍精度の演算とが混在するメソッドを対象として、倍精度演算を行うコード領域を孤立させることにより、第１、第２の実施の形態によるコード生成の効果をさらに向上させる。

本実施の形態では、処理対象であるメソッドのコード解析を行ってメソッド中のコードを単精度演算だけを含む領域（単精度領域）とそれ以外の領域とに分け、倍精度による演算が必要なコード領域を孤立させる。この処理を精度領域解析（PRA; precision region analysis）と呼ぶ。

経験的に、同一メソッドが単精度演算と倍精度演算の両者を含む場合でも、単精度演算と倍精度演算が入り組んで（例えば交互に）現れることはなく、単精度演算がある程度連続するコード領域が見られる。そこで、本実施の形態は、当該メソッド中のコードを制御フローに従って調べ、単精度演算を含みかつ倍精度演算を含まない連続する単精度領域を探索する。単精度領域が得られたならば、当該単精度領域に対して個別にＣＰＵの精度モードを設定することにより、メソッド単位よりもさらに細かいレベルで精度モードの設定を行う。

【００５７】

第４の実施の形態による浮動小数点演算の精度保証のための精度領域解析は、第１の実施の形態と同様に、図１に示すように構成されたコンピュータシステム

にて実現される。

図１６は、本実施の形態におけるコンパイラ１００の機能を説明するブロック図である。

図１６に示すように、本実施の形態のコンパイラ１００は、コンパイル対象のメソッドに対して精度領域解析の処理を行う精度領域解析部１３０と、精度領域解析処理を施されたメソッドのバイナリコードを機械語コードに変換するコード生成部１２０とを備える。精度領域解析部１３０及びコード生成部１２０は、メモリ３００に格納されているプログラムにて制御されたＣＰＵにて実現される仮想的なソフトウェアブロックである。なお図１６には、本実施の形態における特徴的な構成のみを記載している。実際には、図示の構成の他に、コンパイル対象のメソッドのバイナリコードを構文解析する手段や、本実施の形態による過剰精度最適化以外の種々の最適化処理を実行する手段が設けられることはいうまでもない。

【００５８】

精度領域解析部１３０は、コード生成部１２０によるコード生成の前処理として、コンパイル対象のメソッドに対し、制御フローにしたがってコード解析を行い、可能な限り大きな範囲となるように単精度領域を探索する。そして、得られた単精度領域に対して個別にＣＰＵの精度モードの設定を行う。具体的には、各単精度領域に対し、浮動小数点演算における精度保証を行う方法として、ＣＰＵの精度モードを切り替える方法（方法１）またはメモリへの書き込み読み出しを用いて演算精度を落とす方法（方法２）のどちらを用いた場合に処理コストが低いかを調べる。そして、方法１を取った方が処理コストが低いならば、当該単精度領域の入り口と出口で精度モード切り替えを行い、当該単精度領域の処理を単精度モードで実行することとする。一方、方法２を取った方が処理コストが低いならば、当該単精度領域の処理を倍精度モードで実行することとする。

【００５９】

図１７、１８は、精度領域解析部１３０が所定のメソッドに関して、単精度領域を探索し、浮動小数点計算の精度モードを決定する処理の流れを説明するフローチャートである。

図１７、１８に示すように、精度領域解析部１３０は、まず、処理対象として、異なる精度の演算が混在するメソッドのコード（バイナリコード）を入力する。そして、制御フロー上の基本ブロック（制御フローが途中に入ることもなく、途中から出ることもないようなコード列の範囲）を、実行頻度を考慮した深さ優先順により順序づけ、順序Ｏとする（ステップ１７０１）。例えば、図１９（Ａ）に示すソースコードを考えると、これに対して図１９（Ｂ）に示す４つの基本ブロックができ、深さ優先で順序づけると、基本ブロック１、基本ブロック２、基本ブロック４、基本ブロック３の順となる。

【００６０】

また、メソッドの先頭を領域（単精度領域またはその他の領域）Ｒの開始位置とし、当該メソッドが呼ばれた際の精度（例えば、第１の実施の形態では実際にプログラムを走らせて決定された精度モード、第２の実施の形態ではメソッドごとに最適化されるように決定された精度モード）を現在の精度Ｐ（領域Ｒに設定される精度モード）とする（ステップ１７０２）。領域Ｒ及びその精度Ｐは、例えば図２０に示すような管理テーブル（データ構造）を用意し、これに登録して管理する。この管理テーブル２００１は、コンパイラ１００の精度領域解析部１３０にて生成され、メモリ３００に保持される。図２０に示すように、管理テーブル２００１には、領域Ｒを識別するためのＲ番号、開始位置、終了位置、領域内に含まれる基本ブロック、属性（精度モード）といった項目が設けられ、領域Ｒごとに登録される。

【００６１】

次に、精度領域解析部１３０は、順序Ｏにしたがって基本ブロック中の未処理のコードをスキャンする（ステップ１７０３、１７０４、１７０５）。

そして、現在の精度Ｐよりも低い精度（精度Ｐが倍精度である場合に単精度）の計算を行うコードが見つかったならば、現在位置（当該コードの前まで）を領域Ｒの終了位置とする。そして、現在位置（当該コード）を新たに領域Ｒの開始位置として（ステップ１７０６、１７０７）、ステップ１７０３に戻る。

【００６２】

一方、現在の精度Ｐよりも高い精度（精度Ｐが単精度である場合に倍精度）の

計算を行うコードが見つかったならば、上述した方法１による領域Ｒの入り口と出口とで精度切り替えを行う場合の処理コストＣｓと、方法２による領域Ｒに含まれる精度Ｐの計算のそれぞれに対して必要な丸めの処理コストＣｒとを比較する（ステップ１７０８、１７０９）。そして、処理コストＣｓの方が大きい場合は、この領域Ｒを高精度（倍精度）モードで実行するとして、管理テーブル２００１に登録する。また、処理コストＣｒの方が大きい場合は、この領域Ｒを低精度（単精度）モードで実行するとして、管理テーブル２００１に登録する。そして、領域Ｒ内の基本ブロックの切れ目から出ている制御フローの宛先に、決定された精度モードを伝搬させる（ステップ１７１０）。

この後、精度領域解析部１３０は、検出されたコードを次の領域Ｒの開始位置とし、当該コードの精度を当該次の領域Ｒの精度Ｐとして（ステップ１７１１）、ステップ１７０３に戻る。

【００６３】

基本ブロック中のコードをスキャンして、基本ブロックの切れ目（最後のコード）に到達したならば、精度領域解析部１３０は、当該基本ブロックがループのバックエッジ、すなわち当該基本ブロックからループの先頭へ戻る制御フローを持つかどうかを調べる（図１８、ステップ１７１２）。

当該基本ブロックがループのバックエッジを持つならば、このループの先頭は処理中の基本ブロックよりも先に処理されていることが順序Ｏにより保証されている。このループ先頭が属する既に処理された領域Ｒ’を、仮に領域Ｒに含める（ステップ１７１３）。当該基本ブロックがループのバックエッジを持たないならば、次に精度領域解析部１３０は、当該基本ブロックの切れ目から他の基本ブロックへの制御フローがあるかどうかを調べる（ステップ１７１４）。

例えば、図１９に示した例において、深さ優先で基本ブロック１、基本ブロック２、基本ブロック４、基本ブロック３の順でコードのスキャンが行われると、基本ブロック４から基本ブロック３への切れ目には制御フローがなく、その他の基本ブロックの切れ目には制御フローがある。

他の基本ブロックへの制御フローがある場合は、ステップ１７０３に戻り、制御フローの宛先の基本ブロックへ移行し、コードのスキャンを継続する。

【００６４】

一方、他の基本ブロックへの制御フローがない場合、及び基本ブロックがループのバックエッジを持つ場合は、次に、上述した方法１による領域Ｒの入り口と出口とで精度切り替えを行う場合の処理コストＣｓと、方法２による領域Ｒに含まれる精度Ｐの計算のそれぞれに対して必要な丸めの処理コストＣｒとを比較する（ステップ１７１５）。そして、処理コストＣｓの方が大きい場合は、この領域Ｒを高精度（倍精度）モードで実行するとして、管理テーブル２００１に登録する。また、処理コストＣｒの方が大きい場合は、この領域Ｒを低精度（単精度）モードで実行するとして、管理テーブル２００１に登録する。そして、領域Ｒ内の基本ブロックの切れ目から出ている制御フローの宛先に、決定された精度モードを伝搬させる（ステップ１７１６）。

この後、精度領域解析部１３０は、伝搬された精度モードを現在の精度とし、スキャンの開始位置を次の基本ブロックの先頭として（ステップ１７１７）、ステップ１７０３に戻る。

【００６５】

精度領域解析部１３０は、コンパイル対象であるメソッドの全てのコードに対して以上の処理を繰り返し、未処理のコードがなくなったならば、処理を終了する（図１７、ステップ１７０４）

コード生成部１２０は、精度領域解析部１３０にて生成されメモリ３００に保持されている管理テーブル２００１を参照し、領域Ｒごとに精度モードを選択しながら機械語コードを生成する。この結果として、単精度モードを選択された領域Ｒが上述した単精度領域であり、その他の領域に対しては倍精度モードが選択されることとなる。

【００６６】

以上のように、本実施の形態では、単精度の演算と倍精度の演算とが混在するメソッドに対して、コード解析を行ってメソッド中のコードを単精度領域とそれ以外の領域とに分け、倍精度モードが必要なコード領域を孤立させた上で、各々最適な精度モードを基準とするコード生成を行うため、例えば第１の実施の形態と共に本実施の形態を用いれば、倍精度で演算する領域を孤立させ、単精度領域

を単精度モードで扱えるようにすることにより、単精度用の特殊化されたコードの生成対象となるメソッドが多くなる。また、第２の実施の形態と共に本実施の形態を用いることにより、メソッド単位よりもさらに細かいレベルでプログラムの部分ごとに最適なコード生成を行うことが可能となる。

【００６７】

上述した各実施の形態は、浮動小数点演算の演算精度として倍精度と単精度とが用いられている場合について説明した。この他、浮動小数点演算の演算精度として拡張精度が用いられる場合がある。この場合においても、上述した各実施の形態をこの拡張精度にまで適用することができる。例えば、第１の実施の形態においては、基準となる精度モード（例えば倍精度モード）に対して、実際にプログラムを走らせて得られる実績（実行プロファイル）に基づいて、拡張精度モードに特殊化された機械語コードを生成することができる。また、第２の実施の形態においては、コールサイトが静的バインドされることを条件に、コンパイルの対象であるメソッド単体で最適化するように、必要ならば拡張精度モードでコード生成を行う。さらに、第４の実施の形態においては、メソッドに対するコード解析によって、拡張精度が連続するコード領域を孤立させ、適切なコード生成を行うことが可能である。

【００６８】

【発明の効果】

以上説明したように、本発明によれば、浮動小数点演算における単精度演算と倍精度演算の精度保証を効率的に行い、冗長な処理によるオーバーヘッドの発生を回避することが可能となる。

【図面の簡単な説明】

【図１】　第１の実施の形態による浮動小数点演算の精度保証方法が実現されるコンピュータシステムの構成を説明する図である。

【図２】　第１の実施の形態による操作を実現するための、ランタイムルーチンにより生成されるデータ構造を示す図であり、初期状態におけるデータ構造を示す図である。

【図３】　第１の実施の形態による操作を実現するための、ランタイムルー

チンにより生成されるデータ構造を示す図であり、最初のコンパイル時の典型的なデータ構造を示す図である。

【図４】　第１の実施の形態による操作を実現するための、ランタイムルーチンにより生成されるデータ構造を示す図であり、特殊化が実行された場合のデータ構造を示す図である。

【図５】　従来のランタイムルーチンにより生成されるデータ構造を示す図である。

【図６】　第１の実施の形態によるプログラムの特殊化の行程を示すフローチャートである。

【図７】　第２の実施の形態において、コンパイル対象メソッドの呼び出し側が既にコンパイルされている場合のコード生成を説明する図である。

【図８】　第２の実施の形態において、コンパイル対象メソッドの呼び出し側が未だコンパイルされていない場合のコード生成を説明する図である。

【図９】　第２の実施の形態によるコンパイルの手順を説明するフローチャートである。

【図１０】　第３の実施の形態におけるコンパイラの機能を説明する図である。

【図１１】　第３の実施の形態にて用いられる過剰精度計算表を示す図である。

【図１２】　過剰精度最適化部が所定の計算に関して過剰精度か否かを判定する処理の流れを説明するフローチャートである。

【図１３】　過剰精度計算と判定された計算を低精度の演算に置き換える処理の流れを説明するフローチャートである。

【図１４】　第３の実施の形態にて具体的な浮動小数点計算に関して過剰精度計算である倍精度計算を単精度計算に変換する例を示す図であり、単精度浮動小数点値に値０．５を乗算する倍精度計算を単精度化する例を疑似コードで示す図である。

【図１５】　第３の実施の形態にて具体的な浮動小数点計算に関して過剰精度計算である倍精度計算を単精度計算に変換する例を示す図であり、単精度浮動

小数点値の平方根sqrt(v)を倍精度で求めて単精度に丸める計算を単精度化する例を疑似コードで示す図である。

【図１６】　第４の実施の形態におけるコンパイラの機能を説明する図である。

【図１７】　第４の実施の形態において、精度領域解析部が所定のメソッドに関して、単精度領域を探索し、浮動小数点計算の精度モードを決定する処理の流れを説明するフローチャートである。

【図１８】　第４の実施の形態において、精度領域解析部が所定のメソッドに関して、単精度領域を探索し、浮動小数点計算の精度モードを決定する処理の流れを説明するフローチャートである。

【図１９】　所定のソースコードにおける基本ブロックを示す図である。

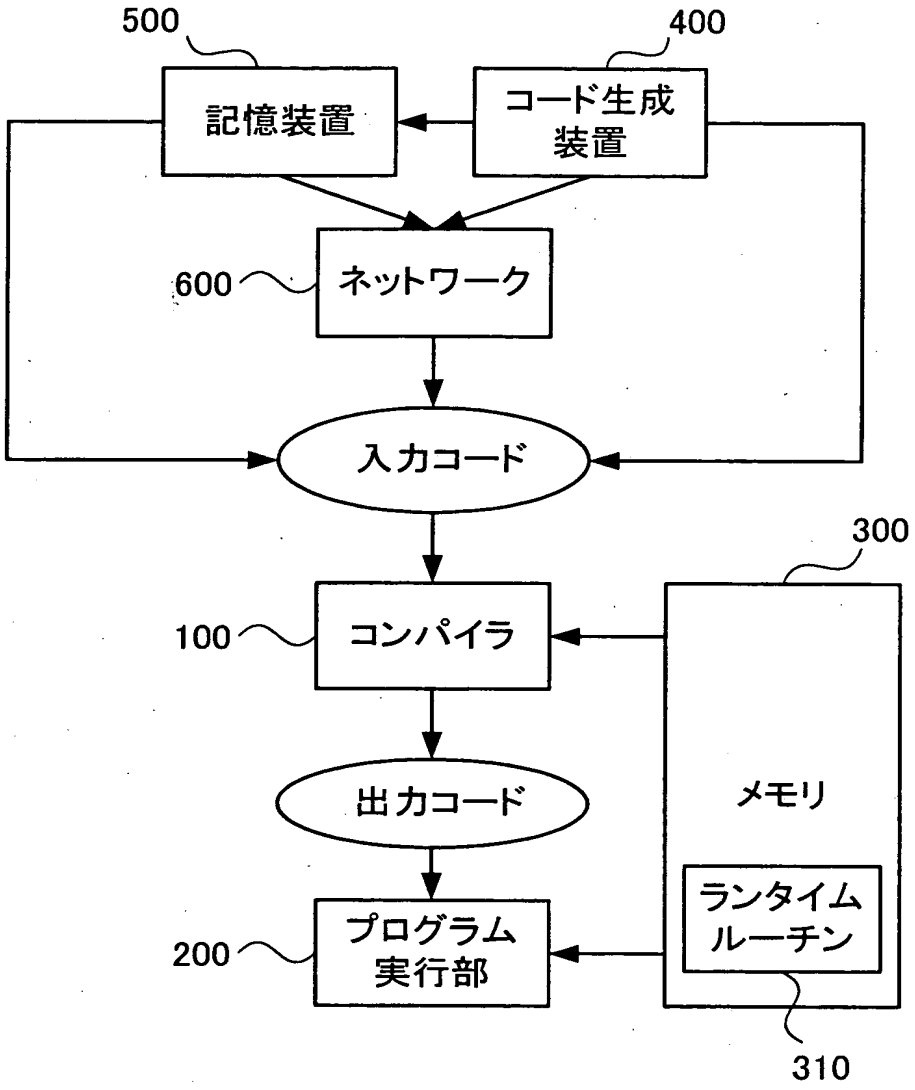
【図２０】　第４の実施の形態にて用いられる管理テーブルの例を示す図である。

【符号の説明】

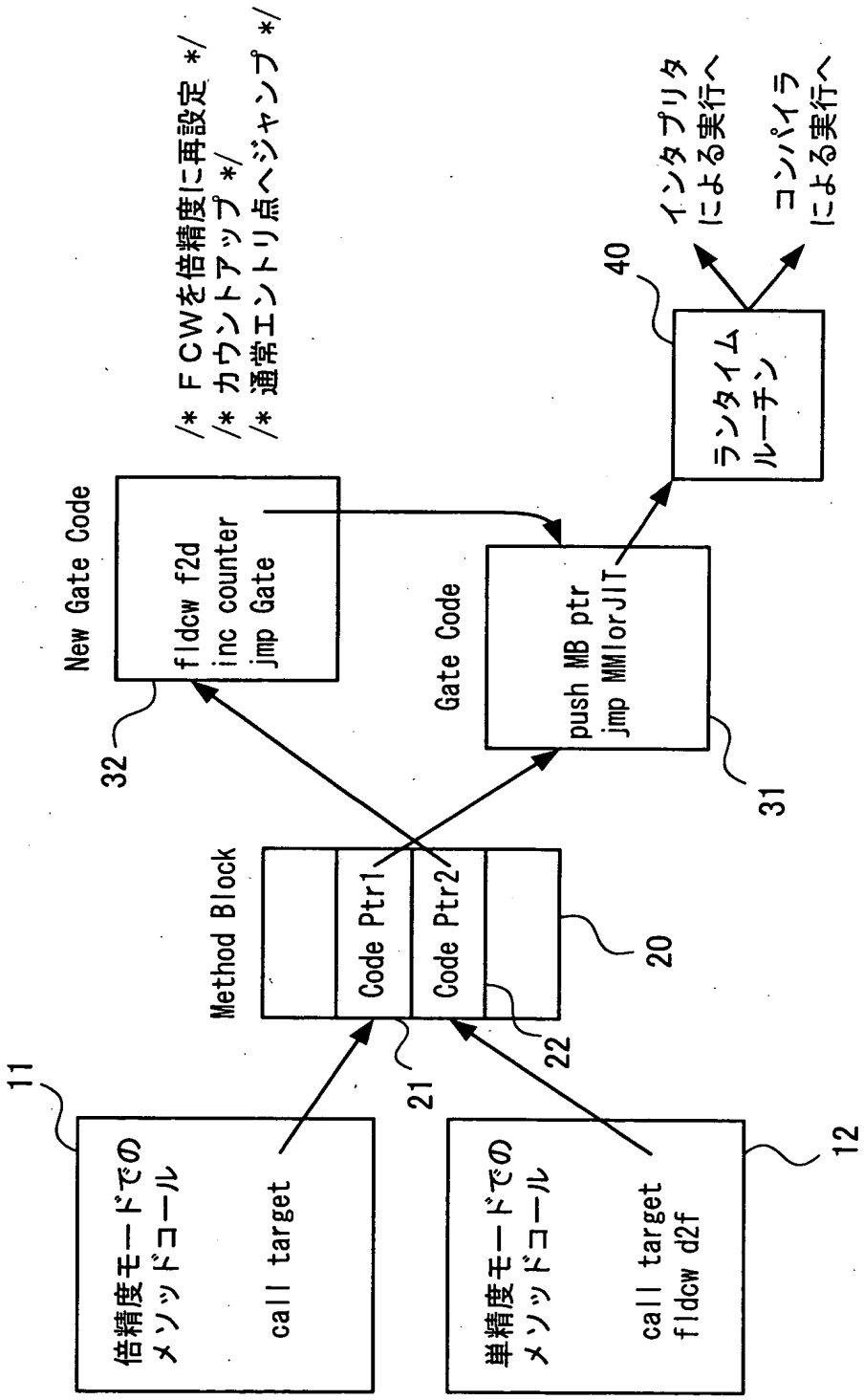
１００…コンパイラ、１１０…過剰精度最適化部、１２０…コード生成部、１３０…精度領域解析部、２００…プログラム実行部、３００…メモリ、３１０…ランタイムルーチン、３２０…過剰精度計算表、４００…コード生成装置、５００…記憶装置、６００…ネットワーク

【書類名】　　図面

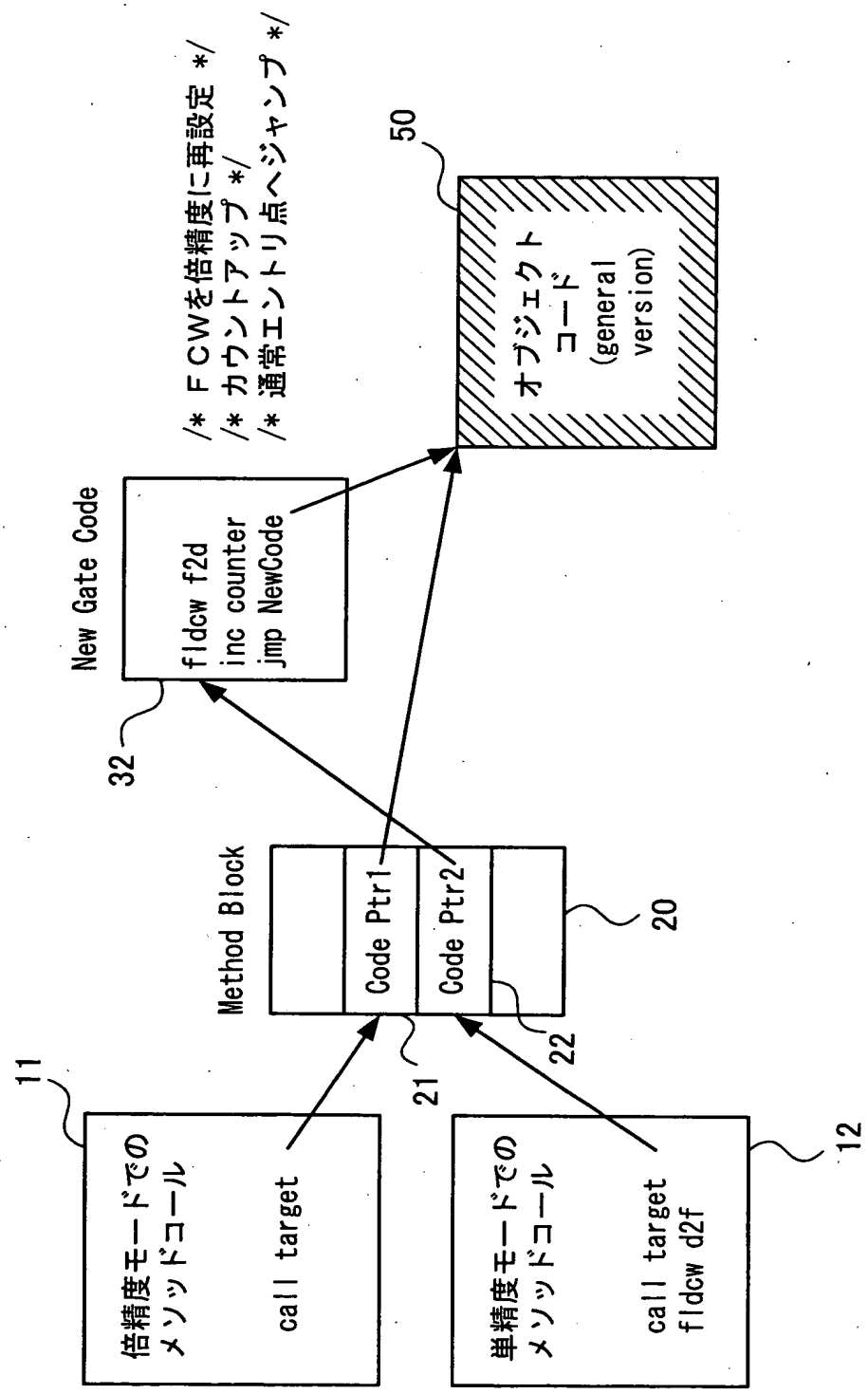
【図１】


500
400
記憶装置
コード生成
装置
600
ネットワーク
入力コード
300
100
コンパイラ
メモリ
出力コード
ランタイム
ルーチン
200
プログラム
実行部
310

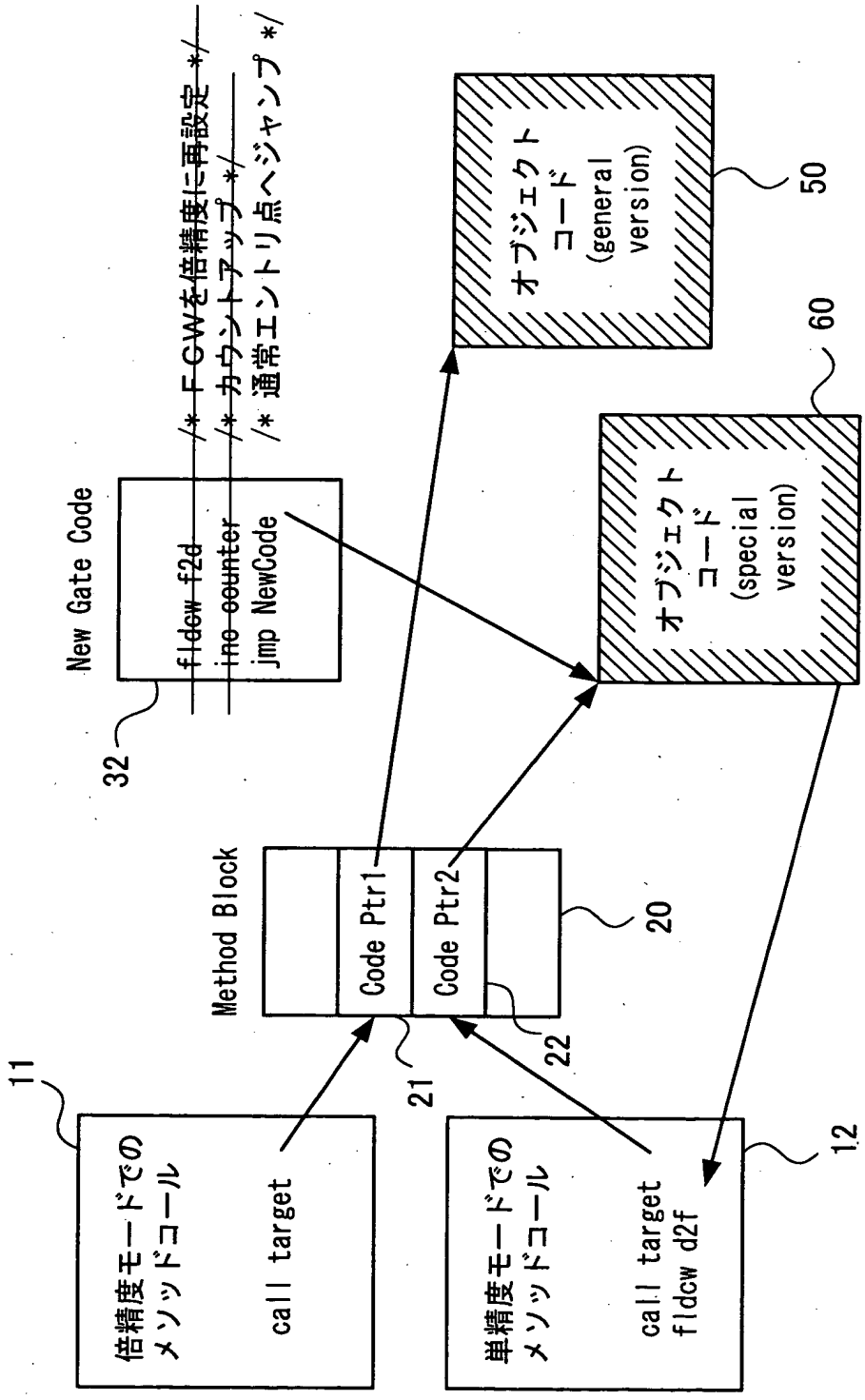
【図２】


倍精度モードでの
メソッドコール
call target
11
単精度モードでの
メソッドコール
call target
fldcw d2f
12
Method Block
Code Ptr1
Code Ptr2
20
21
22
New Gate Code
fldcw f2d
inc counter
jmp Gate
32
/* ＦＣＷを倍精度に再設定 */
/* カウントアップ */
/* 通常エントリ点へジャンプ */
Gate Code
push MB ptr
jmp MMIorJIT
31
ランタイム
ルーチン
40
インタプリタ
による実行へ
コンパイラ
による実行へ

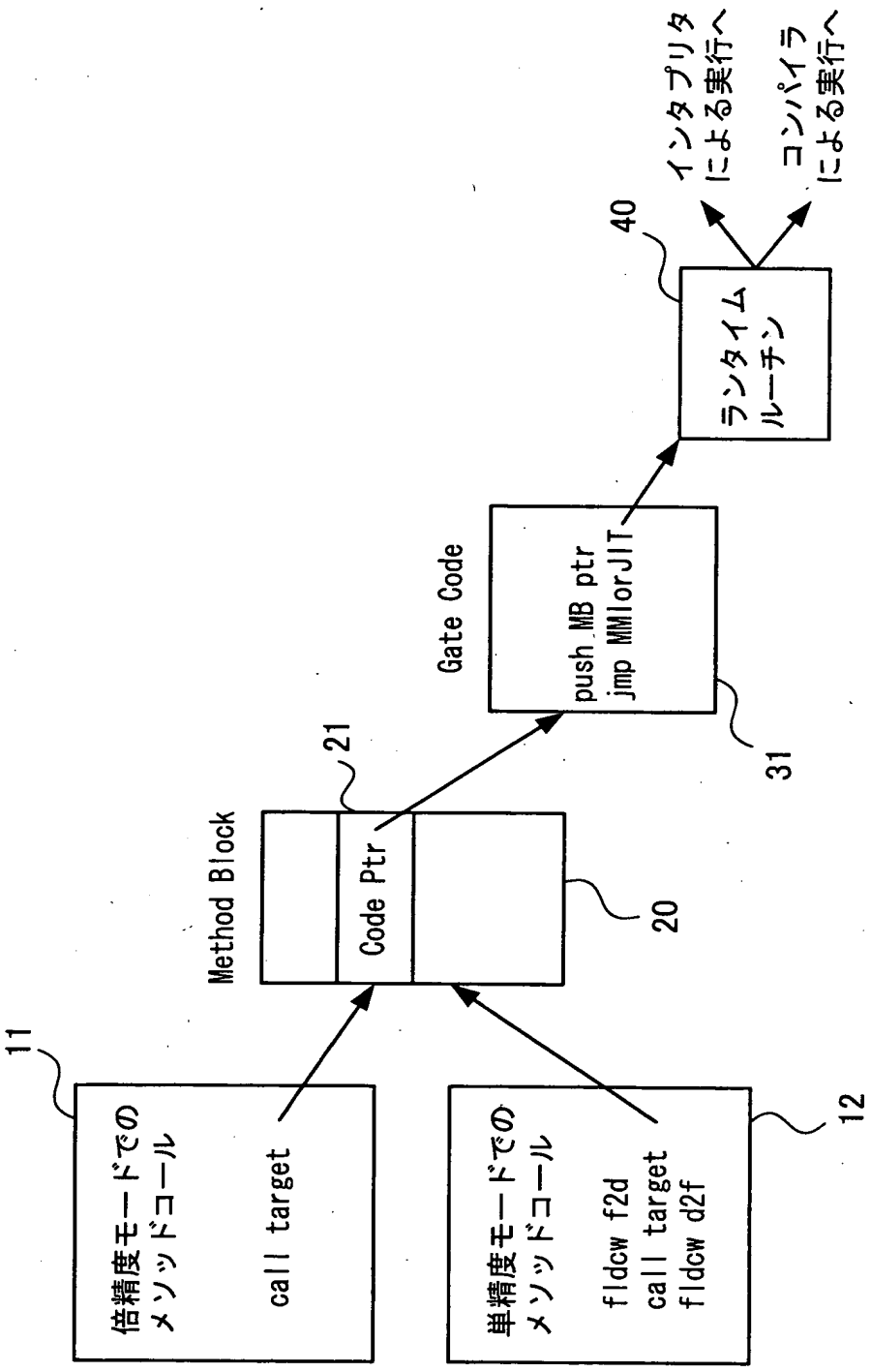
【図３】


倍精度モードでの
メソッドコール
call target
11
単精度モードでの
メソッドコール
call target
fldcw d2f
12
Method Block
Code Ptr1
Code Ptr2
20
21
22
New Gate Code
fldcw f2d
inc counter
jmp NewCode
32
/* ＦＣＷを倍精度に再設定 */
/* カウントアップ */
/* 通常エントリ点へジャンプ */
オブジェクト
コード
(general
version)
50

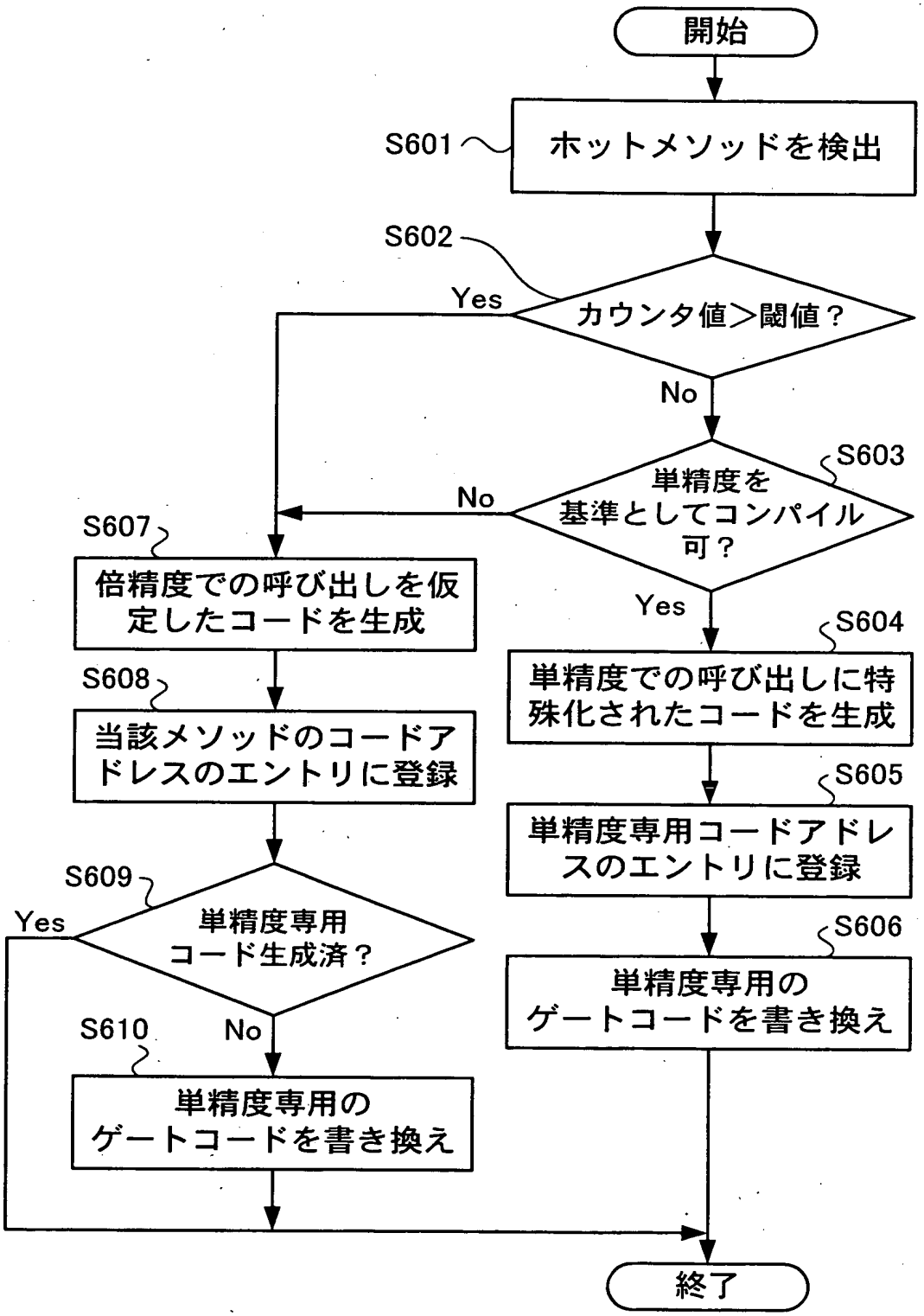
【図４】


倍精度モードでのメソッドコール
call target
11
単精度モードでのメソッドコール
call target
fldcw d2f
12
Method Block
Code Ptr1
Code Ptr2
20
21
22
New Gate Code
fldcw f2d
inc counter
jmp NewCode
/* FCWを倍精度に再設定 */
/* カウントアップ */
/* 通常エントリ点へジャンプ */
32
オブジェクトコード
(general version)
50
オブジェクトコード
(special version)
60

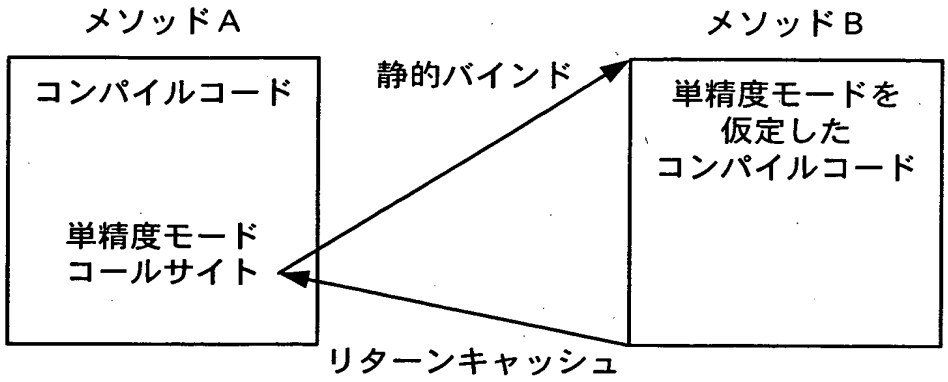
【図５】


倍精度モードでの
メソッドコール
call target
11
単精度モードでの
メソッドコール
fldcw f2d
call target
fldcw d2f
12
Method Block
Code Ptr
21
20
Gate Code
push MB ptr
jmp MMIorJIT
31
ランタイム
ルーチン
40
インタプリタ
による実行へ
コンパイラ
による実行へ

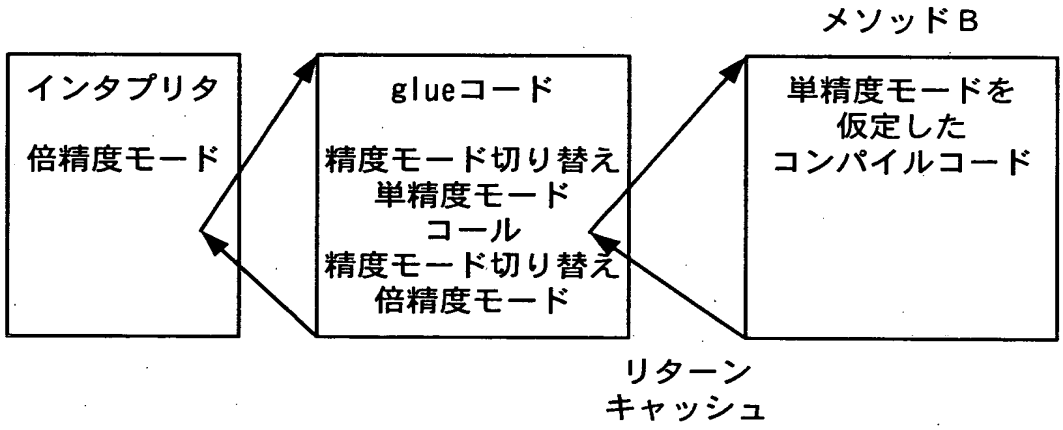
【図６】


開始
S601
ホットメソッドを検出
S602
Yes
カウンタ値＞閾値？
No
S603
単精度を
基準としてコンパイル
可？
No
Yes
S607
倍精度での呼び出しを仮
定したコードを生成
S608
当該メソッドのコードア
ドレスのエントリに登録
S609
Yes
単精度専用
コード生成済？
S610
No
単精度専用の
ゲートコードを書き換え
S604
単精度での呼び出しに特
殊化されたコードを生成
S605
単精度専用コードアドレ
スのエントリに登録
S606
単精度専用の
ゲートコードを書き換え
終了

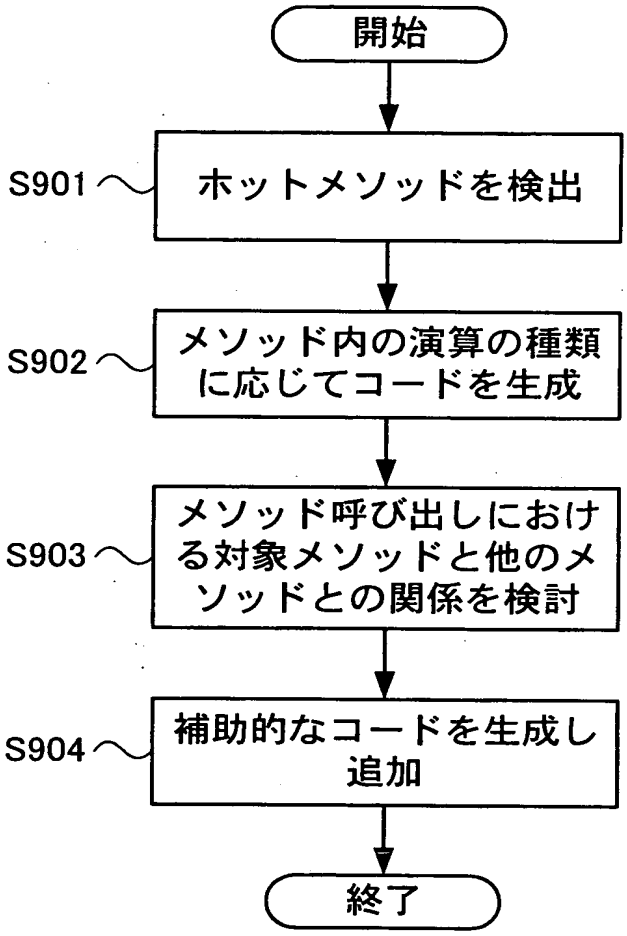
【図７】


メソッドA
メソッドB
コンパイルコード
静的バインド
単精度モードを
仮定した
コンパイルコード
単精度モード
コールサイト
リターンキャッシュ

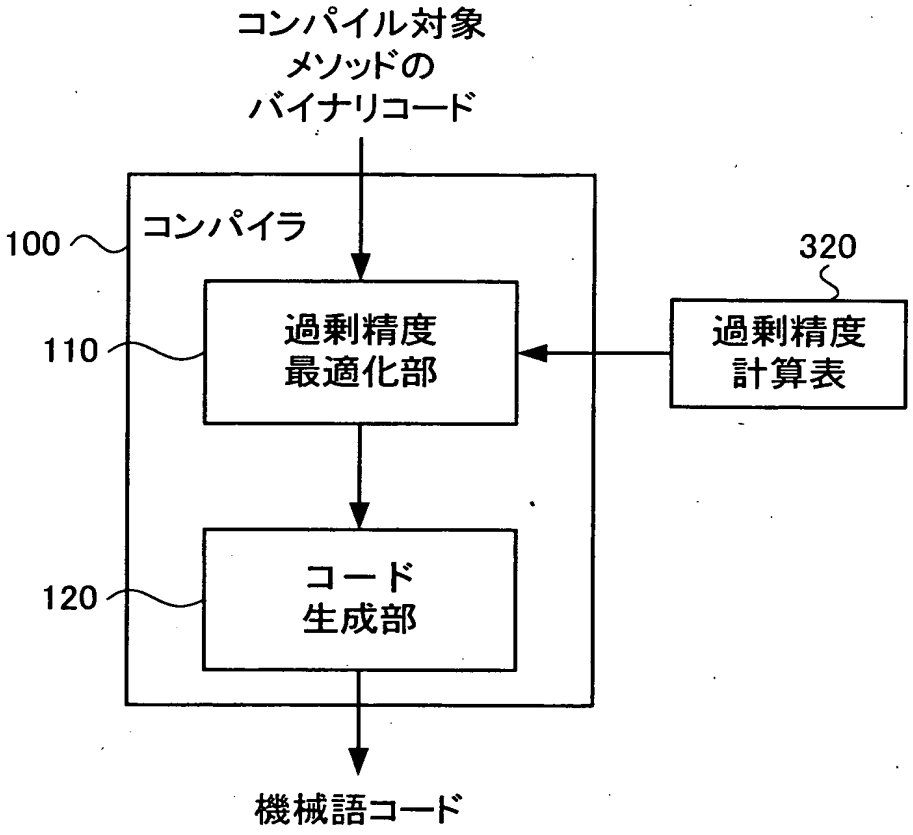
【図８】


メソッドB
インタプリタ
倍精度モード
glueコード
精度モード切り替え
単精度モード
コール
精度モード切り替え
倍精度モード
単精度モードを
仮定した
コンパイルコード
リターン
キャッシュ

【図９】


開始
S901
ホットメソッドを検出
S902
メソッド内の演算の種類に応じてコードを生成
S903
メソッド呼び出しにおける対象メソッドと他のメソッドとの関係を検討
S904
補助的なコードを生成し追加
終了

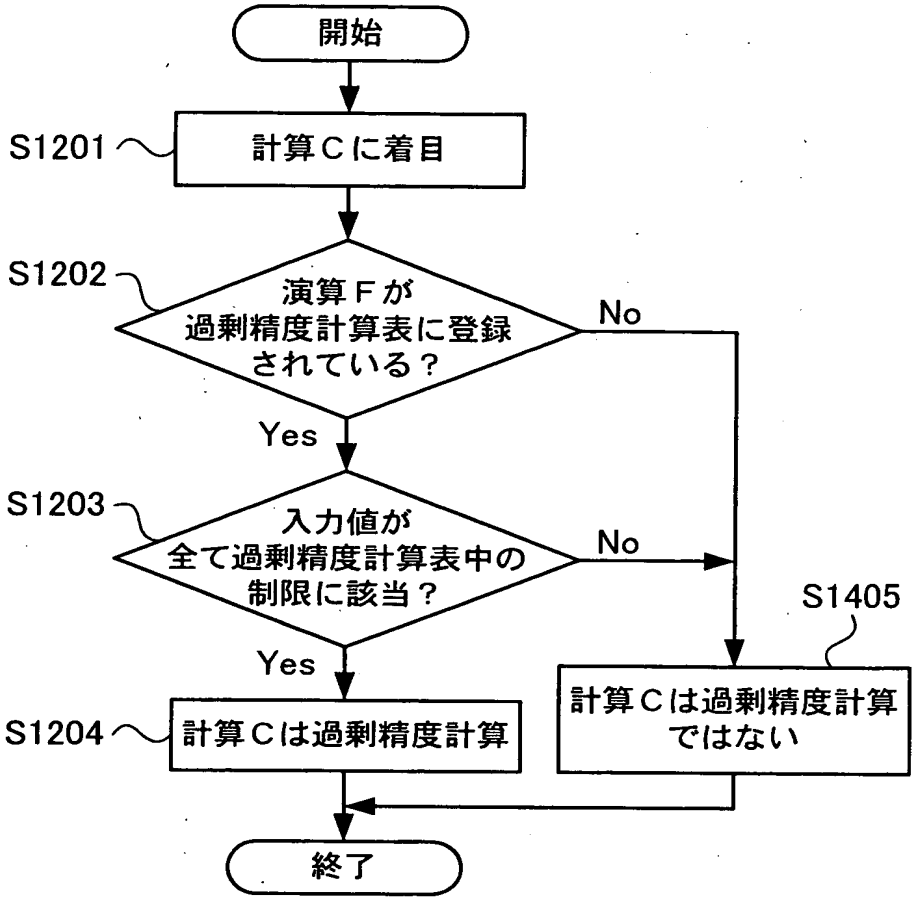
【図１０】



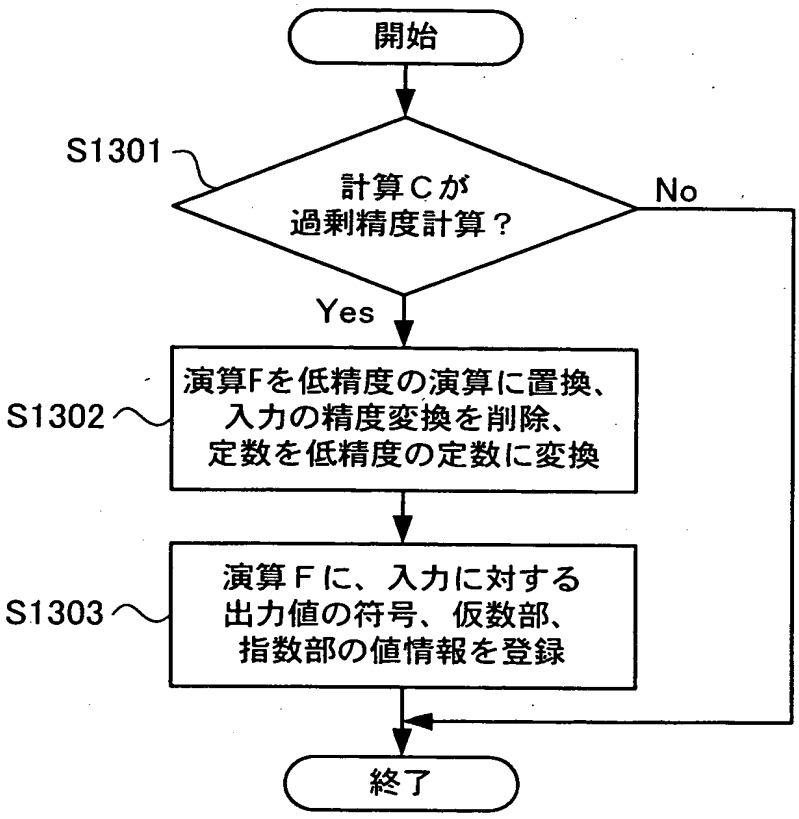
【図１１】

計算種別	入力1の制限	入力2の制限	・・・
DMUL	仮数部が単精度	仮数部が1	
DSQRT+D2F	仮数部が単精度	N.A.	
・ ・ ・			

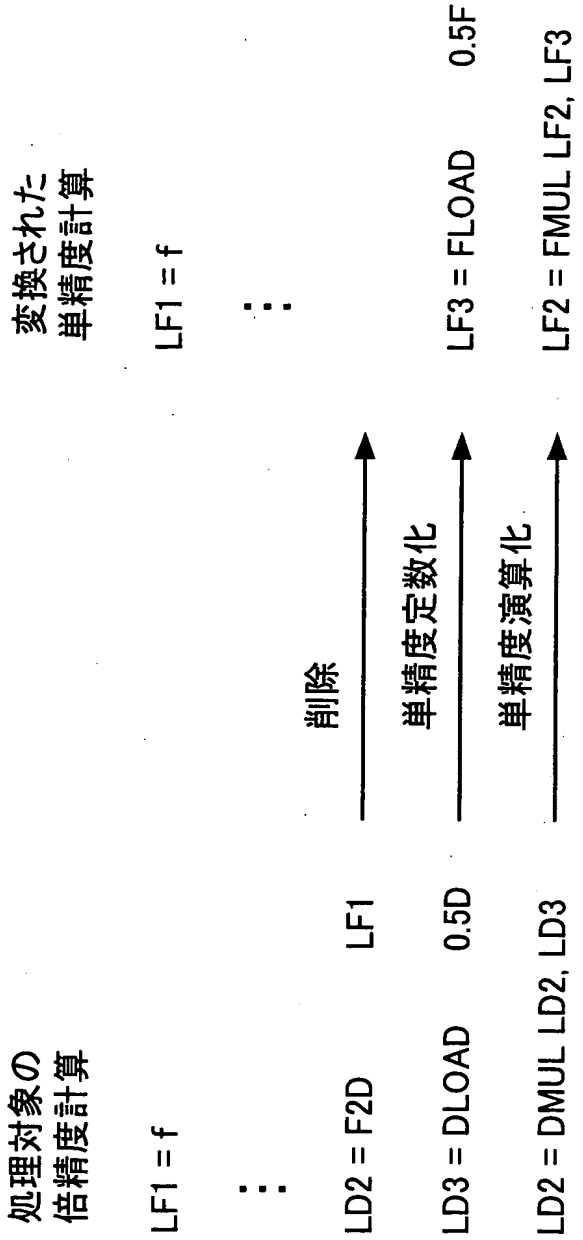
【図１２】


開始
S1201
計算Ｃに着目
S1202
演算Ｆが
過剰精度計算表に登録
されている？
No
Yes
S1203
入力値が
全て過剰精度計算表中の
制限に該当？
No
Yes
S1405
計算Ｃは過剰精度計算
ではない
S1204
計算Ｃは過剰精度計算
終了

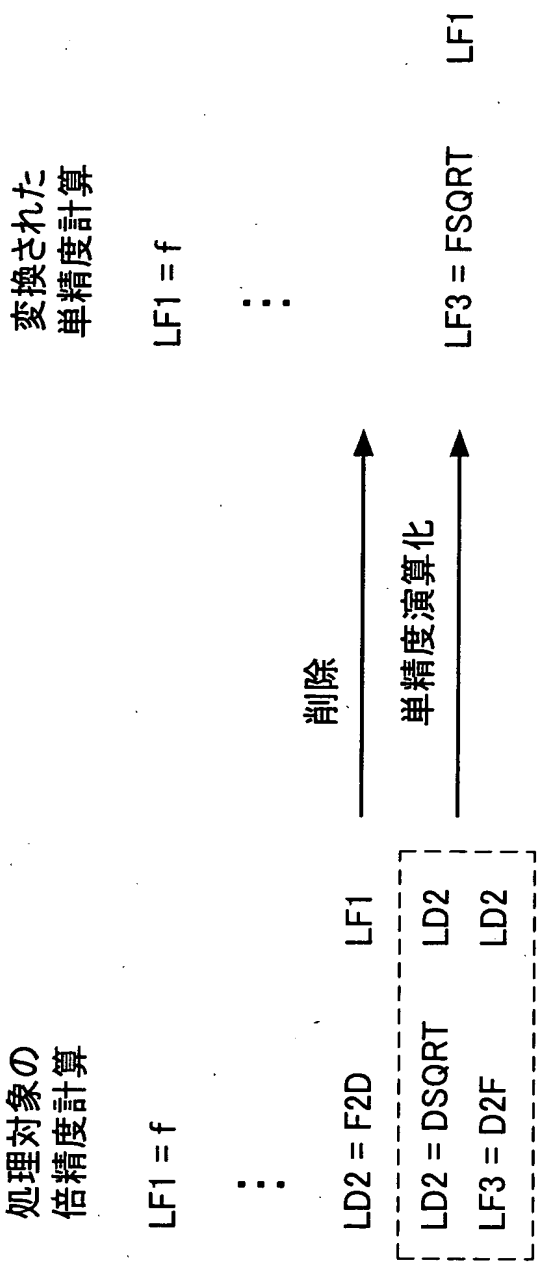
【図１３】


開始
S1301
計算Ｃが
過剰精度計算？
No
Yes
S1302
演算Fを低精度の演算に置換、
入力の精度変換を削除、
定数を低精度の定数に変換
S1303
演算Ｆに、入力に対する
出力値の符号、仮数部、
指数部の値情報を登録
終了

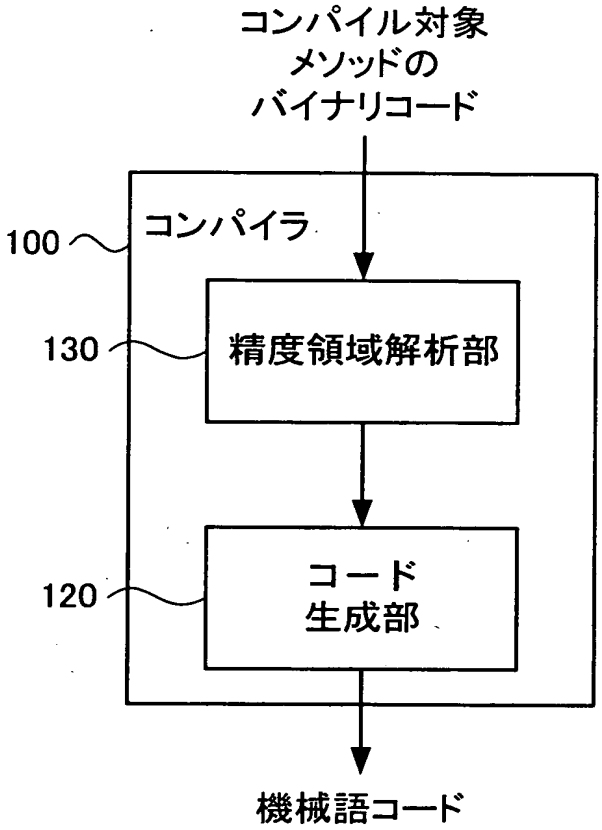
【図１４】


処理対象の
倍精度計算
LF1 = f
：
LD2 = F2D LF1
LD3 = DLOAD 0.5D
LD2 = DMUL LD2, LD3
削除
単精度定数化
単精度演算化
変換された
単精度計算
LF1 = f
：
LF3 = FLOAD 0.5F
LF2 = FMUL LF2, LF3

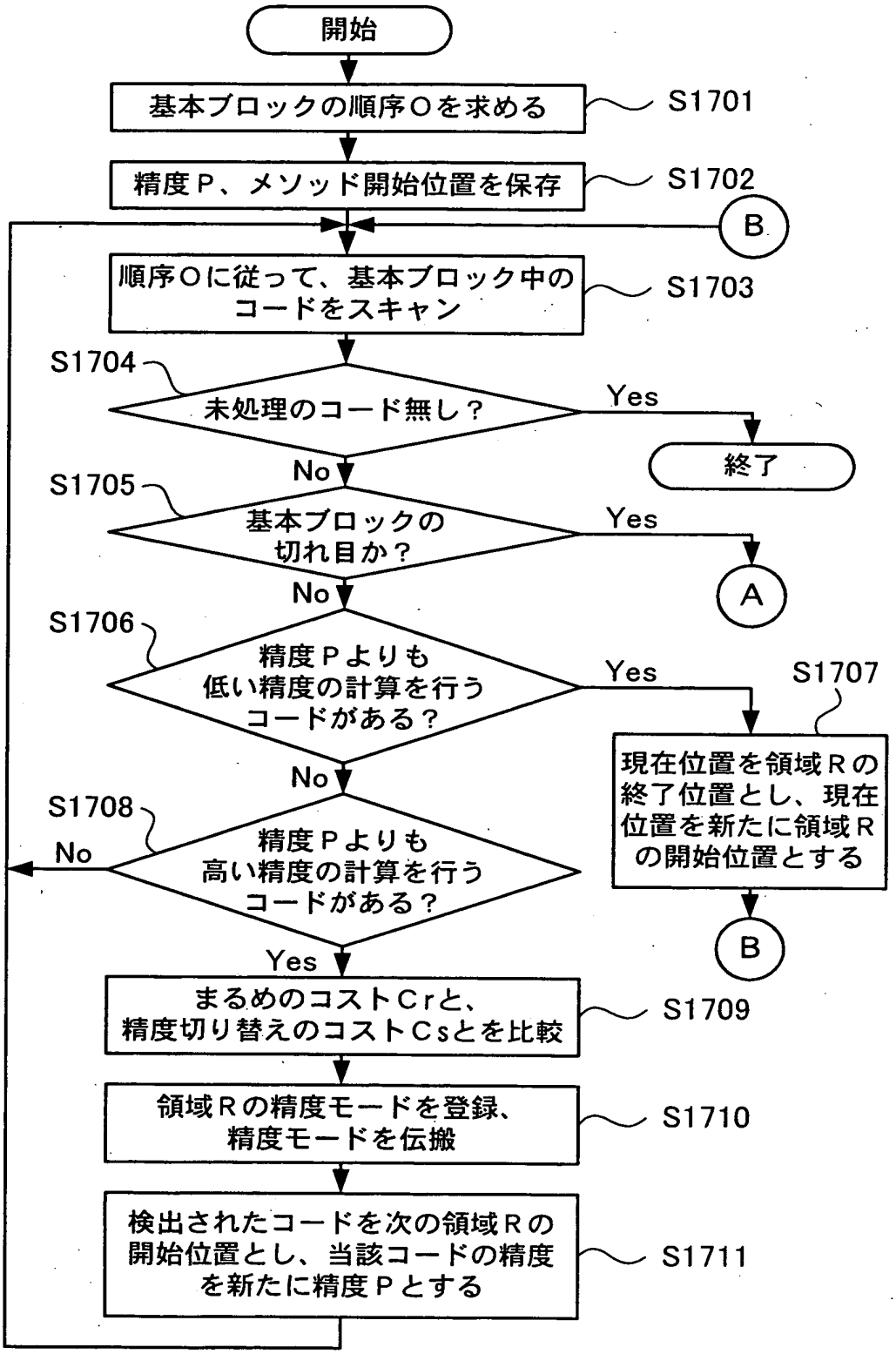
【図１５】


処理対象の
倍精度計算
LF1 = f
：
LD2 = F2D
LF1
LD2 = DSQRT
LD2
LF3 = D2F
LD2
削除
単精度演算化
変換された
単精度計算
LF1 = f
：
LF3 = FSQRT
LF1

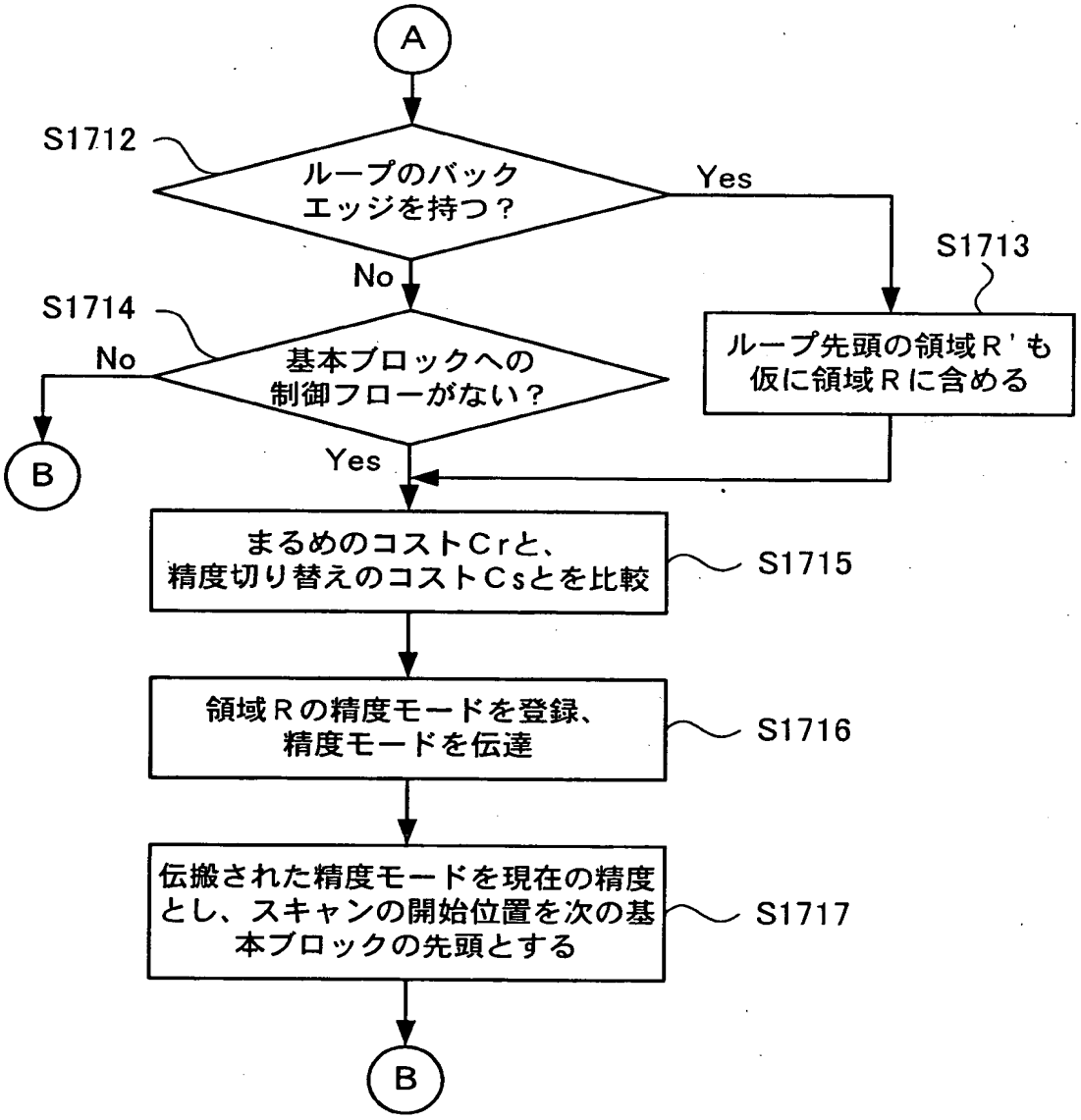
【図１６】


コンパイル対象
メソッドの
バイナリコード
100
コンパイラ
130
精度領域解析部
120
コード
生成部
機械語コード

【図１７】


開始
基本ブロックの順序Oを求める
S1701
精度P、メソッド開始位置を保存
S1702
B
順序Oに従って、基本ブロック中のコードをスキャン
S1703
S1704
未処理のコード無し？
Yes
終了
No
S1705
基本ブロックの切れ目か？
Yes
A
No
S1706
精度Pよりも低い精度の計算を行うコードがある？
Yes
S1707
現在位置を領域Rの終了位置とし、現在位置を新たに領域Rの開始位置とする
B
No
S1708
No
精度Pよりも高い精度の計算を行うコードがある？
Yes
まるめのコストCrと、精度切り替えのコストCsとを比較
S1709
領域Rの精度モードを登録、精度モードを伝搬
S1710
検出されたコードを次の領域Rの開始位置とし、当該コードの精度を新たに精度Pとする
S1711

【図１８】


A
S1712
ループのバック
エッジを持つ？
Yes
No
S1713
ループ先頭の領域Ｒ'も
仮に領域Ｒに含める
S1714
No
基本ブロックへの
制御フローがない？
B
Yes
まるめのコストＣｒと、
精度切り替えのコストＣｓとを比較
S1715
領域Ｒの精度モードを登録、
精度モードを伝達
S1716
伝搬された精度モードを現在の精度
とし、スキャンの開始位置を次の基
本ブロックの先頭とする
S1717
B

【図１９】

（Ａ）

```
if (cond) {
    statement11
    statement12
} else {
    statement21
    statement22
}
statement3
```

（Ｂ）

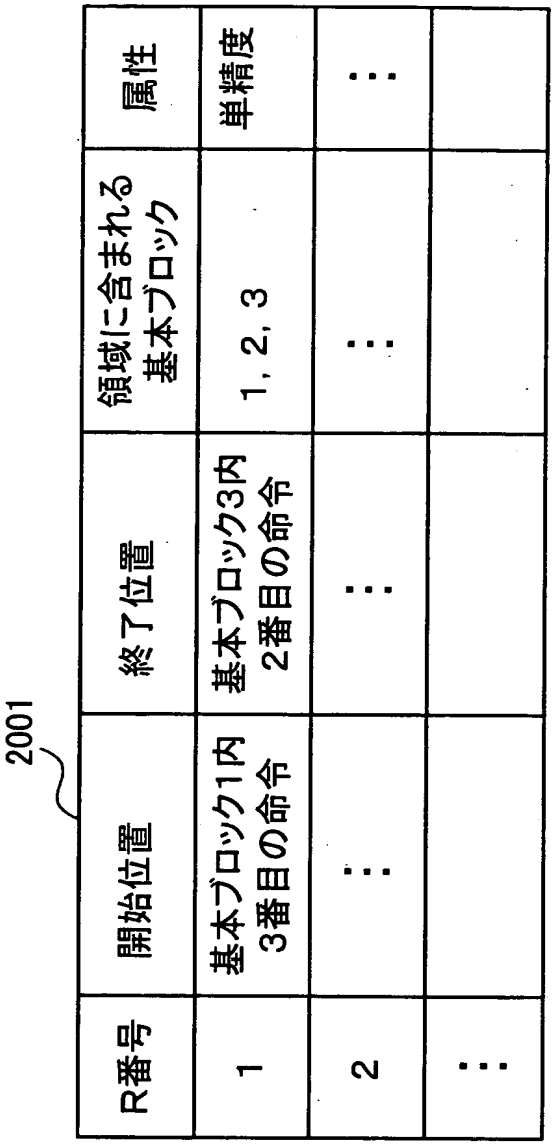
```
基本ブロック１
        test cond
        if true goto 基本ブロック２
        goto 基本ブロック３

基本ブロック２
        statement11
        statement12
        goto 基本ブロック４

基本ブロック３
        statement21
        statement22
        goto 基本ブロック４

基本ブロック４
        statement3
```

【図２０】


2001
R番号
開始位置
終了位置
領域に含まれる基本ブロック
属性
1
基本ブロック1内3番目の命令
基本ブロック3内2番目の命令
1, 2, 3
単精度
2

【書類名】　　要約書

【要約】

【課題】　浮動小数点演算における単精度演算と倍精度演算の精度保証を効率的に行い、オーバーヘッドや解析処理の無駄の発生を回避する。

【解決手段】　コンピュータに搭載されたコンパイラ１００において、浮動小数点演算における倍精度モードでメソッドのオブジェクトコードを生成すると共に、このメソッドが単精度モードで頻繁に呼び出され、かつ単精度モードで実行することにより処理コストを削減できる場合に、単精度モードでメソッドの第２のコードを生成する。あるいは、コンパイルしようとする対象メソッドが浮動小数点演算における倍精度モードと単精度モードのどちらから呼ばれるかに関わらず、この対象メソッド内の演算の種類に応じて演算精度を設定してオブジェクトコードを生成し、対象コードとこの対象コードを呼び出す他のコードまたはこの対象コードに呼び出される他のコードとの関係に基づいて精度モードを合わせるためのコードを生成し追加する。

【選択図】　　図１

認定・付加情報

特許出願の番号	特願２００２－２２４２０７
受付番号	５０２０１１３６１９３
書類名	特許願
担当官	佐々木　吉正　　２４２４
作成日	平成１４年　９月　９日

＜認定情報・付加情報＞

【特許出願人】
- 【識別番号】 390009531
- 【住所又は居所】 アメリカ合衆国１０５０４、ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
- 【氏名又は名称】 インターナショナル・ビジネス・マシーンズ・コーポレーション

【代理人】
- 【識別番号】 100086243
- 【住所又は居所】 神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　大和事業所内
- 【氏名又は名称】 坂口　博

【代理人】
- 【識別番号】 100091568
- 【住所又は居所】 神奈川県大和市下鶴間１６２３番地１４　日本アイ・ビー・エム株式会社　大和事業所内
- 【氏名又は名称】 市位　嘉宏

【代理人】
- 【識別番号】 100108501
- 【住所又は居所】 神奈川県大和市下鶴間１６２３番１４　日本アイ・ビー・エム株式会社　知的所有権
- 【氏名又は名称】 上野　剛史

【復代理人】 申請人
- 【識別番号】 100104880
- 【住所又は居所】 東京都港区赤坂５－４－１１　山口建設第２ビル６Ｆ　セリオ国際特許事務所
- 【氏名又は名称】 古部　次郎

次頁無

出　願　人　履　歴　情　報

識別番号　　〔３９００００９５３１〕

１．変更年月日　　２００２年　６月　３日
［変更理由］　住所変更
住　所　　アメリカ合衆国１０５０４、ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
氏　名　　インターナショナル・ビジネス・マシーンズ・コーポレーション